```
       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 Widget Controller - EPROM firmware
       8/       0 :                     ;                        Version 341-0289-D (1A45)
       9/       0 :                     ;            original code (c) by Apple Computer Inc., 1983
      10/       0 :                     ;          'Widget-1O: System-Code: Rev:1-A.4.5 Dec 31, 1983'
      11/       0 :                     ;
      12/       0 :                     ;
      13/       0 :                     ;                    disassembly by Al Kossow, 2011
      14/       0 :                     ;  commented and converted into as source by Patrick Schaefer, 2011
      15/       0 :                     ;   revised and merged with Apple's original V1-A.4.4 source, 2013
      16/       0 :                     ;
      17/       0 :                     ;    use as build 2011-08-02 or later. Last changes: 2013-07-31 PS
      18/       0 :                     ;
      19/       0 :                     ;*****************************************************************
      20/       0 :
      21/       0 : =1H                 DontListIncls   SET 1                   ; set to skip listing for .inc
      22/       0 :
      23/       0 : =1H                 W_10MB          SET 1                   ; choose your disk capacity
      24/       0 : =0H                 W_20MB          SET 0
      25/       0 : =0H                 W_40MB          SET 0
      26/       0 : =0H                 DevSubType      SET 0                   ; 0 = System, 1 = Diag Firmware
      27/       0 :
      28/       0 :                     ; 1A44
      29/       0 :                     ; HiRevNumber   SET 01Ah                ; not used here but good to know
      30/       0 :                     ; LoRevNumber   SET 044h
      31/       0 : =1AH                HiRevNumber     SET 01Ah                ; not used here but good to know
      32/       0 : =45H                LoRevNumber     SET 045h
      33/       0 : =2000H              ROMsize         SET 8192
      34/       0 :                     ; 1A44
      35/       0 :                     ; Checksum0     SET 0D4FBh              ; for ROM test
      36/       0 :                     ; Checksum1     SET 0EF07h
      37/       0 : =D47CH              Checksum0       SET 0D47Ch              ; for ROM test
      38/       0 : =EF78H              Checksum1       SET 0EF78h
      39/       0 : =F078H              Passwrd1        SET 0F078h              ; upper nibble high shifted right
      40/       0 : =3C1EH              Passwrd2        SET 03C1Eh
      41/       0 :
      42/       0 :
      43/       0 :                     ; this source contains Bank0.Assem (1000h), Format.Assem (10C5h), Cmnd.Assem
      44/       0 :                     ; (1195h), Spare.Assem (1353h), Spare1.Assem (146Eh), Cache.Assem (163Ah),
      45/       0 :                     ; Data.X.Assem (16B1h), SprUtils.Assem (1786h), Cmnd0.Assem (1818h),
      46/       0 :                     ; Cmnd1.Assem (1999h), Cmnd2.Assem (1C74h), Write.Assem (1DB8h), Read.Assem
      47/       0 :                     ; (1EA9h), and Srvo0.Assem (1F8Dh) in the lower 4 kB.
      48/       0 :                     ;
      49/       0 :                     ; Bank1.Assem (2000h), RdHdr.B1.Assem (2007h), Utils.B1.Assem (21EAh),
      50/       0 :                     ; Spr1.B1.Assem (229Fh), Math.B1.Assem (2565h), Srvo1.B1.Assem (2605h),
      51/       0 :                     ; Srvo2.B1.Assem (2837h), SlfTst.B1.Assem (2A22h), Cache.B1.Assem (2B41h),
      52/       0 :                     ; Progs.B1.Assem (2CC1h), Ecc.B1.Assem (2F06h) are located in the upper 4kB.
      53/       0 :
      54/       0 :                             include DefsWidget.inc          ; Defs1.Assem, Defs2.Assem
(1)    1/       0 :                                     save
(1)    2/       0 : =>TRUE                      if DontListIncls
(1)  905/    174E :                                     restore
(1)  906/    174E :
(1)  907/    174E :
(1)  908/    174E :
      55/    174E :                             include vectors0288.inc         ; addresses referred to in Z8 ROM
(1)    1/    174E :                                     save
(1)    2/    174E : =>TRUE                      if DontListIncls
(1)   52/    174E :                                     restore
(1)   53/    174E :
      56/    174E :
      57/    174E :
      58/    174E :                     ;*****************************************************************
      59/    174E :                     ; Module Bank0.Assem
      60/    174E :                     ;
      61/    174E :                     ;*****************************************************************
      62/       0 :                             org 0000h
      63/    1000 :                             phase 1000h             ; EPROM will start at 4k
      64/    1000 :                             assume RP:Wrk_Sys
      65/    1000 : D4 7C                               DW Checksum0
      66/    1002 : 00                                  DB 0                    ; bank 0
      67/    1003 : F0 78 3C 1E      PassWord           DW Passwrd1, Passwrd2
      68/    1007 : 1F               FmtDelay           DB 01Fh                 ; busy wait for a while
      69/    1008 : 01 01            RWI_Value          DW RWI_Cylinder         ; cylinder at which to turn on RWI and PC
      70/    100A :                     ;
      71/    100A :                     ; bank 0 vector table
      72/    100A :                     B0_VctTab
      73/    100A : 8D 11 95            Start_Vector    jp Start_Command
      74/    100D : 8D 06 0D            LL_Vector       jp Load_Logical
      75/    1010 : 8D 05 E1            LH_Vector       jp Load_Header
      76/    1013 : 8D 12 CA            RdL_Vector      jp Rd_Leave
      77/    1016 : 8D 1E 5F            CS_Vector       jp ClrNormStat
      78/    1019 : 2C 25               LdPw_Vector     ld R2, #Load_PassWord /256
      79/    101B : 3C 06                               ld R3, #Load_PassWord #256
      80/    101D : D6 04 AB                            call Bank_Call
      81/    1020 : AF                                  ret
      82/    1021 : 2C 2E               Free_Vector     ld R2, #Strt_FreeProcess /256
      83/    1023 : 3C 42                               ld R3, #Strt_FreeProcess #256
      84/    1025 : D6 04 AB                            call Bank_Call
      85/    1028 : AF                                  ret
      86/    1029 : 2C 22               ExtStk_Vector   ld R2, #Init_ExtStack /256
      87/    102B : 3C 04                               ld R3, #Init_ExtStack #256
      88/    102D : D6 04 AB                            call Bank_Call
      89/    1030 : AF                                  ret
      90/    1031 : 2C 20               ZrRd_Vector     ld R2, #Zero_RdBuf /256
      91/    1033 : 3C B6                               ld R3, #Zero_RdBuf #256
      92/    1035 : D6 04 AB                            call Bank_Call
      93/    1038 : AF                                  ret
      94/    1039 : 2C 06               ClrStat_Vector  ld R2, #ClearStatus /256
      95/    103B : 3C 3D                               ld R3, #ClearStatus #256
      96/    103D : D6 04 AB                            call Bank_Call
      97/    1040 : AF                                  ret
      98/    1041 : 2C 2A               SlfTst_Vector   ld R2, #SelfTest /256
      99/    1043 : 3C 22                               ld R3, #SelfTest #256
     100/    1045 : D6 04 AB                            call Bank_Call
     101/    1048 : AF                                  ret
```

```
102/    1049 : 2C 23              SprTbl_Vector   ld R2, #Load_SprTbl /256
103/    104B : 3C 1D                              ld R3, #Load_SprTbl #256
104/    104D : D6 04 AB                           call Bank_Call
105/    1050 : AF                                 ret
106/    1051 : 2C 2B              LC_Vector       ld R2, #Load_Cache /256
107/    1053 : 3C 41                              ld R3, #Load_Cache #256
108/    1055 : D6 04 AB                           call Bank_Call
109/    1058 : AF                                 ret
110/    1059 : 2C 2C              Scan_Vector     ld R2, #D_Scan /256
111/    105B : 3C C1                              ld R3, #D_Scan #256
112/    105D : D6 04 AB                           call Bank_Call
113/    1060 : 8D 12 CA                           jp Rd_Leave
114/    1063 : 0C 00              IScan_Vector:   ld R0, #DevSubType      ; check for SystemCode
115/    1065 : 42 00                              or R0, R0
116/    1067 : EB 13                              jr NZ, IScan_Ret
117/    1069 : 76 24 10                           tm Excpt_Stat, #PwrRst  ; do scan only after power reset
118/    106C : 6B 07                              jr Z, IScan_SprChk
119/    106E : 2C 2C                              ld R2, #Scan /256
120/    1070 : 3C CA                              ld R3, #Scan #256
121/    1072 : D6 04 AB                           call Bank_Call
122/    1075 : 2C 25              IScan_SprChk    ld R2, #Chk_SprCnt /256
123/    1077 : 3C 4B                              ld R3, #Chk_SprCnt #256
124/    1079 : D6 04 AB                           call Bank_Call
125/    107C : AF                 IScan_Ret       ret
126/    107D : D6 1D B8           WrBlk_Vector    call WriteBlock
127/    1080 : 8D 04 F8           VctrB0_Ret      jp Bank_Ret
128/    1083 : D6 1E A9           RdBlk_Vector    call ReadBlock
129/    1086 : 8B F8                              jr VctrB0_Ret
130/    1088 : D6 16 5E           SC_Vector       call SrchCache
131/    108B : 8B F3                              jr VctrB0_Ret
132/    108D : 2C 26              Seek_Vector     ld R2, #Seek /256
133/    108F : 3C 05                              ld R3, #Seek #256
134/    1091 : D6 04 AB                           call Bank_Call
135/    1094 : 8D 04 F8                           jp Bank_Ret
136/    1097 : 2C 22              ExtPush_Vector  ld R2, #Ext_Push /256
137/    1099 : 3C 15                              ld R3, #Ext_Push #256
138/    109B : D6 04 AB                           call Bank_Call
139/    109E : AF                                 ret
140/    109F : 2C 22              ExtPop_Vector   ld R2, #Ext_Pop /256
141/    10A1 : 3C 36                              ld R3, #Ext_Pop #256
142/    10A3 : D6 04 AB                           call Bank_Call
143/    10A6 : AF                                 ret
144/    10A7 :                    ;
145/    10A7 :                    DeviceParams
146/    10A7 : =>TRUE                     if W_10MB
147/    10A7 : 57 69 64 67 65 74                  DB "Widget-10    "
               2D 31 30 20 20 20
               20
148/    10B4 : 00 01 00                           DB 00, 01, 000h+DevSubType
149/    10B7 : 1A 45                              DB HiRevNumber, LoRevNumber
150/    10B9 : 00 4C 00                           DB 000h, 04Ch, 000h     ; number of logical blocks
151/    10BC : 02 14                              DW 532                  ; number of bytes per block
152/    10BE : 02 02                              DW NbrTracks            ; number of cylinders is 514
153/    10C0 : 02                                 DB NbrHds               ; number of heads is 2
154/    10C1 : 13                                 DB NbrSctrs             ; NbrSctrs
155/    10C2 : 00 00 4C                           DB 0, 0, 76             ; number of spare possible
156/    10C5 : [146]                      endif
157/    10C5 : =>FALSE                    if W_20MB
158/    10C5 :                                    DB "Widget-20    "
159/    10C5 :                                    DB 00, 01, 010h+DevSubType
160/    10C5 :                                    DB HiRevNumber, LoRevNumber
161/    10C5 :                                    DB 000h, 098h, 000h     ; number of logical blocks
162/    10C5 :                                    DW 532                  ; number of bytes per block
163/    10C5 :                                    DW NbrTracks            ; number of cylinders is 514
164/    10C5 :                                    DB NbrHds               ; number of heads is 2
165/    10C5 :                                    DB NbrSctrs             ; NbrSctrs
166/    10C5 :                                    DB 0, 0, 76             ; number of spare possible
167/    10C5 : [157]                      endif
168/    10C5 : =>FALSE                    if W_40MB
169/    10C5 :                                    DB "Widget-40    "
170/    10C5 :                                    DB 00, 01, 020h+DevSubType
171/    10C5 :                                    DB HiRevNumber, LoRevNumber
172/    10C5 :                                    DB 001h, 030h, 000h     ; number of logical blocks
173/    10C5 :                                    DW 532                  ; number of bytes per block
174/    10C5 :                                    DW NbrTracks            ; number of cylinders is 514
175/    10C5 :                                    DB NbrHds               ; number of heads is 2
176/    10C5 :                                    DB NbrSctrs             ; NbrSctrs
177/    10C5 :                                    DB 0, 0, 76             ; number of spare possible
178/    10C5 : [168]                      endif
179/    10C5 : =1EH               Dev_Parm_Length EQU $ - DeviceParams
180/    10C5 :
181/    10C5 :
182/    10C5 :
183/    10C5 :                    ;*****************************************************************
184/    10C5 :                    ; Module Format.Assem
185/    10C5 :                    ;
186/    10C5 :                    ; This module contains all but the very most primitive of
187/    10C5 :                    ; procedures that concern themselves with performing a format
188/    10C5 :                    ; operation.
189/    10C5 :                    ;
190/    10C5 :                    ;       PROCEDURE FormatTrack(Offset, InterLeave)
191/    10C5 :                    ;       PROCEDURE LocateSector
192/    10C5 :                    ;
193/    10C5 :                    ;*****************************************************************
194/    10C5 :
195/    10C5 :                    ;*****************************************************************
196/    10C5 :                    ;
197/    10C5 :                    ; Procedure: FormatTrack
198/    10C5 :                    ;
199/    10C5 :                    ;  This function is responsible for formatting the track at which
200/    10C5 :                    ;  the heads are currently positioned.
201/    10C5 :                    ;
202/    10C5 :                    ; Inputs: Offset:     BYTE (R4)
203/    10C5 :                    ;         InterLeave: BYTE (R5)
204/    10C5 :                    ;
205/    10C5 :                    ; Outputs: none
206/    10C5 :                    ;
207/    10C5 :                    ; Algorithm:
208/    10C5 :                    ;
209/    10C5 :                    ;  Begin
210/    10C5 :                    ;   For i:=1 To Length(FormatArray) Do
211/    10C5 :                    ;     FormatArray[i]:=0
212/    10C5 :                    ;   While not(index) Do Begin End
213/    10C5 :                    ;   Turn on AC erase {FmenL}
214/    10C5 :                    ;   If IndexMark
215/    10C5 :                    ;     Then
216/    10C5 :                    ;       Turn off AC erase
217/    10C5 :                    ;       While (Offset>0) Do
```

```
218/    10C5 :                      ;         While not(SectorMark) Do Begin End
219/    10C5 :                      ;         While SectorMark Do Begin End
220/    10C5 :                      ;         Offset:=Offset-1
221/    10C5 :                      ;       InterleaveFactor:=InterLeaveTable[InterLeave]
222/    10C5 :                      ;       Sector:=0
223/    10C5 :                      ;       For i:=1 To NbrSctrs Do
224/    10C5 :                      ;         If not(FormatBlock) Then Abort
225/    10C5 :                      ;         Sector:=(Sector+InterLeaveFactor) mod NbrSctrs
226/    10C5 :                      ;  End
227/    10C5 :                      ;
228/    10C5 :                      ;*********************************************************************
229/    10C5 :
230/    10C5 : =>TRUE                       if W_10MB
231/    10C5 : 02 01 07 0A 08 0D    InterTable      DB 2, 1, 7, 10, 8, 13, 3        ; interleave table
               03
232/    10CC : =>FALSE                      elseif
233/    10CC :                      InterTable      DB 2, 1, 26, 10, 8, 13, 22      ; interleave table
234/    10CC : [230]                        endif
235/    10CC :
236/    10CC : 2C 20                FormatTrack     Ld R2,#Zero_Fmt /256
237/    10CE : 3C B8                                Ld R3,#Zero_Fmt #256
238/    10D0 : D6 04 AB                             Call Bank_Call
239/    10D3 : 46 56 20                             Or DiskStat,#Wr_Op
240/    10D6 :                      ; do one revolution AC erase
241/    10D6 : 76 03 04             FmtT_1          tm P3, #IndexMark       ; while not index
242/    10D9 : 6B FB                                jr Z, FmtT_1
243/    10DB : 56 00 DF                             and P0, #0FFh-AcEraseL  ; turn on AC erase
244/    10DE : 76 03 04             FmtT_2          tm P3, #IndexMark       ; while index
245/    10E1 : EB FB                                jr NZ, FmtT_2
246/    10E3 : 76 03 04             FmtT_3          tm P3, #IndexMark       ; while not index
247/    10E6 : 6B FB                                jr Z, FmtT_3
248/    10E8 : 46 00 20                             or P0, #AcEraseL        ; turn off AC erase
249/    10EB : 42 44                                or R4, R4               ; test for zero offset
250/    10ED : 6B 0A                                jr Z, Fmt_Begin
251/    10EF :                      ; skip the first (offset) sectors
252/    10EF : 56 FA FB             Fmt_Off1        And IRQ, #0FFh-Irq_Sector       ; clear old sector mark
253/    10F2 : 76 FA 04             Fmt_Off2        tm IRQ, #Irq_Sector     ; wait for next mark to pass
254/    10F5 : 6B FB                                jr Z, Fmt_Off2
255/    10F7 : 4A F6                                djnz R4, Fmt_Off1
256/    10F9 :                      ; and here we go
257/    10F9 : 2C 10                Fmt_Begin       ld R2, #InterTable /256
258/    10FB : 3C C5                                ld R3, #InterTable #256
259/    10FD : 02 35                                add R3, R5              ; get offset into table
260/    10FF : 16 E2 00                             adc R2, #0
261/    1102 : C2 62                                ldc R6, @RR2
262/    1104 : B0 55                                clr Sector              ; sector := 0
263/    1106 : 5C 13                                ld R5, #NbrSctrs
264/    1108 : 1C 08                FmtTrk_2        ld R1, #Dmt_FmtTrack
265/    110A : D6 04 46                             call FormatBlock        ; write one sector
266/    110D : EB 05                                jr NZ, FmtT_Until       ; problems?
267/    110F : 98 E0                                ld R9, R0               ; pass reason for abort
268/    1111 : D6 05 1F                             call Abort
269/    1114 : 04 E6 55             FmtT_Until      add Sector, R6          ; sector := sector + InterLeaveFactor
270/    1117 : A6 55 13                             cp Sector, #NbrSctrs    ; sector := sector mod NbrSctrs
271/    111A : 1B 03                                jr LT, FmtT2_Until
272/    111C : 26 55 13                             sub Sector, #NbrSctrs   ; do mod by subtraction
273/    111F : 5A E7                FmtT2_Until     djnz R5, FmtTrk_2
274/    1121 : 8D 04 F8                             jp Bank_Ret
275/    1124 :
276/    1124 :
277/    1124 :                      ;*********************************************************************
278/    1124 :                      ;
279/    1124 :                      ; Procedure: LocateSector
280/    1124 :                      ;
281/    1124 :                      ;  This procedure returns to the caller just after the sector
282/    1124 :                      ;  mark representing the sector JUST BEFORE the sector of the
283/    1124 :                      ;  last seek address.
284/    1124 :                      ;
285/    1124 :                      ; Inputs: none
286/    1124 :                      ;
287/    1124 :                      ; Outputs: none
288/    1124 :                      ;
289/    1124 :                      ; Global Variables Used: Sector
290/    1124 :                      ;
291/    1124 :                      ; Algorithm:
292/    1124 :                      ;
293/    1124 :                      ;  Begin
294/    1124 :                      ;    Offset:=SpareTable.FmtOffset
295/    1124 :                      ;    InterL:=SpareTable.InterLeave
296/    1124 :                      ;    SectorCount:=0
297/    1124 :                      ;    Temp:=0
298/    1124 :                      ;    While not(Index) Do Begin End
299/    1124 :                      ;    While (Offset<>0) Do
300/    1124 :                      ;      Wait for End of Sector Mark
301/    1124 :                      ;    While (Temp mod NbrSctrs <> Sector) Do
302/    1124 :                      ;      Temp:=Temp+InterL
303/    1124 :                      ;      i:=i+2 {count sectors at 2:1 interleave}
304/    1124 :                      ;    While (i<>0) Do
305/    1124 :                      ;      Wait for End of Sector Mark
306/    1124 :                      ;      i:=i+1
307/    1124 :                      ;  End
308/    1124 :                      ;
309/    1124 :                      ;*********************************************************************
310/    1124 :
311/    1124 : 0C 09                LocateSector    ld R0, #Dmt_LctSctr
312/    1126 : D6 04 13                             call Set_Dmt
313/    1129 : 70 FD                                push RP                 ; save context
314/    112B : 31 30                                srp #Wrk_Sys2
315/    112D :                              assume RP:Wrk_Sys2
316/    112D : 2C 14                                ld R2, #FmtOffset /256
317/    112F : 3C C3                                ld R3, #FmtOffset #256
318/    1131 : 82 42                                lde R4, @RR2            ; load offset value
319/    1133 : 44 54 54                             or Head, Head           ; check for even/odd head
320/    1136 : 6B 02                                jr Z, Lct_Offset
321/    1138 : F0 E4                                swap R4
322/    113A : 56 E4 0F             Lct_Offset      and R4, #0Fh
323/    113D : A0 E2                                incw RR2                ; point to interleave value
324/    113F : 82 52                                lde R5, @RR2
325/    1141 : 2C 10                                ld R2, #InterTable /256
326/    1143 : 3C C5                                ld R3, #InterTable #256
327/    1145 : 02 35                                add R3, R5              ; add in interleave value
328/    1147 : 16 E2 00                             adc R2, #0
329/    114A : C2 52                                ldc R5, @RR2            ; get real interleave value
330/    114C :                      ;
331/    114C : 76 03 04             LCtSctr1        tm p3, #IndexMark       ; while index do begin end
332/    114F : EB FB                                jr NZ, LctSctr1
333/    1151 : 76 03 04             LctSctr2        tm p3, #IndexMark       ; while not index do begin end3
334/    1154 : 6B FB                                jr Z, LctSctr2
```

```
335/    1156 : 56 FA FB                         and IRQ, #0FFh-Irq_Sector       ; clear any old sector marks
336/    1159 : 42 44                            or R4, R4               ; if offset = 0 then we're already here
337/    115B : 6B 05                            jr Z, LctSctr4
338/    115D : 28 E4                            ld R2, R4
339/    115F : D6 11 8A                         call LctSctr3
340/    1162 : 44 55 55         LctSctr4        or Sector, Sector       ; if sector = 0 then we're already there
341/    1165 : 6B 1B                            jr Z, LctDone
342/    1167 : B0 E2                            clr R2
343/    1169 : B0 E4                            clr R4
344/    116B : A4 55 E4         LctSctr5        cp R4, Sector           ; see if we've found the sector
345/    116E : 6B 0F                            jr Z, LctSctr6
346/    1170 : 06 E2 02                         add R2, #2
347/    1173 : 02 45                            add R4, R5              ; temp := temp + InterLeave Factor
348/    1175 : A6 E4 13                         cp R4, #NbrSctrs        ; temp := temp mod NbrSctrs
349/    1178 : 1B F1                            jr LT, LctSctr5
350/    117A : 26 E4 13                         sub R4, #NbrSctrs
351/    117D : 8B EC                            jr LctSctr5
352/    117F : D6 11 8A         LctSctr6        call LctSctr3
353/    1182 : D6 04 1F         LctDone         call Clr_Dmt
354/    1185 : 50 FD                            pop RP                  ; restore original context
355/    1187 : 8D 04 F8                         jp Bank_Ret
356/    118A :                  ;
357/    118A : 76 FA 04         LctSctr3        tm IRQ, #Irq_Sector     ; wait for next sector
358/    118D : 6B FB                            jr Z, LctSctr3
359/    118F : 56 FA FB                         and IRQ, #0FFh-Irq_Sector       ; mask old interrupt
360/    1192 : 2A F6                            djnz R2, LctSctr3       ; count the number of sectors in offset
361/    1194 : AF                               ret
362/    1195 :
363/    1195 :
364/    1195 :
365/    1195 :                  ;*****************************************************************
366/    1195 :                  ; Module Cmnd.Assem
367/    1195 :                  ;
368/    1195 :                  ; This module controls the way in which commands received by
369/    1195 :                  ; the Host are executed. Its main responsibility is to determine
370/    1195 :                  ; whether a command string is protected by a check byte (the
371/    1195 :                  ; original ProFile commands are not) and then decode the command
372/    1195 :                  ; and pass control over to the correct driver routine to
373/    1195 :                  ; execute the command. All exception handling at this level is
374/    1195 :                  ; controlled by the individual command routines.
375/    1195 :                  ;
376/    1195 :                  ;       PROCEDURE Start_Command
377/    1195 :                  ;       PROCEDURE Pro_Read
378/    1195 :                  ;       FUNCTION Read_Common : BOOLEAN
379/    1195 :                  ;
380/    1195 :                  ;*****************************************************************
381/    1195 :
382/    1195 :                  ;*****************************************************************
383/    1195 :                  ;
384/    1195 :                  ; Procedure: StartCommand
385/    1195 :                  ;
386/    1195 :                  ;  Assumes a command string in external memory; the command byte
387/    1195 :                  ;  must be in location "Cmnd_Ptr".
388/    1195 :                  ;
389/    1195 :                  ; Algorithm:
390/    1195 :                  ;
391/    1195 :                  ;  Begin
392/    1195 :                  ;    Initialize internal and external stack ptrs
393/    1195 :                  ;    ZeroHeader
394/    1195 :                  ;    Excpt_Stat.Buf_Damage:=false
395/    1195 :                  ;    Excpt_Stat.Nzero_Stat:=false
396/    1195 :                  ;    If Cache_Index = Cahe_Length
397/    1195 :                  ;      Then Cache_Index:=0
398/    1195 :                  ;    OpCode:=ExtMem[0]
399/    1195 :                  ;    If OpCode.CommandType is FreeProcess type
400/    1195 :                  ;      Then goto Strt_FreeProcess
401/    1195 :                  ;    If OpCode.CommandType is protected by a checkbyte
402/    1195 :                  ;      Then
403/    1195 :                  ;        If not(Check_Command_String)
404/    1195 :                  ;          Then Abort(Cmnd_Driver_Exception,
405/    1195 :                  ;                     Start_Command, CheckByte_Mismatch)
406/    1195 :                  ;    If OpCode.Instruction > CommandLimit[CommandType]
407/    1195 :                  ;      Then Abort(Cmnd_Driver_Exception,
408/    1195 :                  ;                 Start_Command, IllegalOpCode, OpCode)
409/    1195 :                  ;    JMP @Command^[CommandType].RoutineTable[Instruction]
410/    1195 :                  ;  End
411/    1195 :                  ;
412/    1195 :                  ;*****************************************************************
413/    1195 :
414/    1195 : 31 10            Start_Command   srp #Wrk_Sys            ; get into a reasonable state
415/    1197 :                          assume RP:Wrk_Sys
416/    1197 : B0 FE                            clr SPH
417/    1199 : E6 FF 80                         ld SPL, #Stack_Top
418/    119C : 2C 17                            ld R2, #StackPtr /256   ; init external stack
419/    119E : 3C 4C                            ld R3, #StackPtr #256
420/    11A0 : 0C 17                            ld R0, #TopOfStack /256
421/    11A2 : 92 02                            lde @RR2, R0
422/    11A4 : A0 E2                            incw RR2
423/    11A6 : 0C FF                            ld R0, #TopOfStack #256
424/    11A8 : 92 02                            lde @RR2, R0
425/    11AA : A6 5B 14                         cp Cache_Index, #CacheLength
426/    11AD : 1B 02                            jr LT, St_Load_Cmnd
427/    11AF : B0 5B                            clr Cache_Index
428/    11B1 : 2C 10            St_Load_Cmnd    ld R2, #Cmnd_Ptr /256
429/    11B3 : 3C 1A                            ld R3, #Cmnd_Ptr #256
430/    11B5 : EC 14                            ld R14, #CStatus4 /256
431/    11B7 : FC A5                            ld R15, #CStatus4 #256
432/    11B9 : 1C 08                            ld R1, #8               ; move 8 bytes
433/    11BB : 82 02            St_L_Lp         lde R0, @RR2
434/    11BD : 92 0E                            lde @RR14, R0
435/    11BF : A0 E2                            incw RR2
436/    11C1 : A0 EE                            incw RR14
437/    11C3 : 1A F6                            djnz R1, St_L_Lp
438/    11C5 :                  ; check for $F0 command which means 'do nothing, go idle'
439/    11C5 : EC 14            Strt_ZH         ld R14, #CStatus4 /256
440/    11C7 : FC A5                            ld R15, #CStatus4 #256
441/    11C9 : 82 4E                            lde R4, @RR14
442/    11CB : 68 E4                            ld R6, R4
443/    11CD : 56 E6 F0                         and R6, #CmndType
444/    11D0 : A6 E6 F0                         cp R6, #0F0h            ; check for 'free bus command'
445/    11D3 : EB 07                            jr NZ, Start_ChkDiag
446/    11D5 : 2C 2E                            ld R2, #Strt_FreeProcess /256
447/    11D7 : 3C 42                            ld R3, #Strt_FreeProcess #256
448/    11D9 : D6 04 AB                         call Bank_Call
449/    11DC :                  ; commands starting with $10 are diagnostic commands
450/    11DC : A6 E6 10         Start_ChkDiag   cp R6, #10h             ; check for diagnostic command
451/    11DF : EB 0A                            jr NZ, Strt_Swap
452/    11E1 : 2C 2A                            ld R2, #Set_SeekNeeded /256     ; invalidate cache
```

```
453/    11E3 : 3C 0E                            ld R3, #Set_SeekNeeded #256
454/    11E5 : D6 04 AB                         call Bank_Call
455/    11E8 : E6 57 90                         ld Seek_Type, #Access_Offset
456/    11EB : F0 E6            Strt_Swap       swap R6                 ; r6 := CommandType
457/    11ED : 6B 15                            jr Z, Chk_Inst          ; jump if no check byte in string
458/    11EF :                  ;
459/    11EF : 88 E4                            ld R8, R4
460/    11F1 : 56 E8 0F                         and R8, #0Fh            ; get length of command string
461/    11F4 : A0 EE                            incw RR14               ; get opcode
462/    11F6 : 82 4E                            lde R4, @RR14
463/    11F8 : EC 10                            ld R14, #Cmnd_Ptr /256
464/    11FA : FC 1A                            ld R15, #Cmnd_Ptr #256
465/    11FC : D6 06 1D                         call Chk_Chk_Byte
466/    11FF : 6B 03                            jr Z, Chk_Inst
467/    1201 : D6 05 1F                         call Abort
468/    1204 :                  ;
469/    1204 : A6 E6 02         Chk_Inst        cp R6, #Max_Cmnd_Types  ; check for illegal type
470/    1207 : AB 1E                            jr GT, Inst_Abort
471/    1209 : 44 25 25                         or SlfTst_Result, SlfTst_Result ; check if we're not healthy
472/    120C : 6B 08                            jr Z, Chk_Limits
473/    120E : A6 E6 01                         cp R6, #1               ; only allow diagnostic commands
474/    1211 : 6B 03                            jr Z, Chk_Limits        ;  when selftest failed
475/    1213 : D6 05 1F                         call Abort
476/    1216 :                  ;
477/    1216 : EC 12            Chk_Limits      ld R14, #Cmnd_Limits /256
478/    1218 : FC 4A                            ld R15, #Cmnd_Limits #256
479/    121A : 02 F6                            add R15, R6             ; get offset into table
480/    121C : 16 EE 00                         adc R14, #0
481/    121F : C2 0E                            ldc R0, @RR14           ; get limit
482/    1221 : 18 E4                            ld R1, R4               ; get instruction
483/    1223 : A2 10                            cp R1, R0
484/    1225 : 2B 03                            jr LE, Ok_Inst          ; jump if legal instruction
485/    1227 :                  ;
486/    1227 : D6 05 1F         Inst_Abort      call Abort
487/    122A :                  ;
488/    122A : EC 12            Ok_Inst         ld R14, #Cmnd_Ptrs /256
489/    122C : FC 4D                            ld R15, #Cmnd_Ptrs #256
490/    122E : 90 E6                            rl R6
491/    1230 : 02 F6                            add R15, R6             ; get offset into table
492/    1232 : 16 EE 00                         adc R14, #0
493/    1235 : C2 CE                            ldc R12, @RR14          ; get inst routine ptr
494/    1237 : A0 EE                            incw RR14
495/    1239 : C2 DE                            ldc R13, @RR14
496/    123B : 90 E1                            rl R1
497/    123D : 02 D1                            add R13, R1             ; get offset into table
498/    123F : 16 EC 00                         adc R12, #0
499/    1242 : C2 EC                            ldc R14, @RR12          ; get address to routine
500/    1244 : A0 EC                            incw RR12
501/    1246 : C2 FC                            ldc R15, @RR12
502/    1248 : 30 EE                            jp @RR14                ; jump to routine
503/    124A :
504/    124A :                  ;*****************************************************************
505/    124A :                  :
506/    124A :                  ; Command Routine Tables
507/    124A :                  ;
508/    124A :                  ;*****************************************************************
509/    124A :
510/    124A : 02               Cmnd_Limits     DB Pro_Cmnds
511/    124B : 13                               DB Diag_Cmnds
512/    124C : 02                               DB Sys_Cmnds
513/    124D :
514/    124D : 12 53            Cmnd_Ptrs       DW Pro_Inst
515/    124F : 12 59                            DW Diag_Inst
516/    1251 : 12 81                            DW Sys_Inst
517/    1253 :
518/    1253 : 12 87            Pro_Inst        DW Pro_Read
519/    1255 : 18 7A                            DW Pro_Write
520/    1257 : 19 42                            DW Pro_WrVer
521/    1259 :
522/    1259 : 18 18            Diag_Inst       DW D_Read_ID            ; cmnd 0
523/    125B : 1A 27                            DW Read_CStatus         ; cmnd 1
524/    125D : 1A B3                            DW Read_SStatus         ; cmnd 2
525/    125F : 1A F7                            DW Send_ServoCmnd       ; cmnd 3
526/    1261 : 1B 22                            DW Send_Seek            ; cmnd 4
527/    1263 : 1B 46                            DW Send_Restore         ; cmnd 5
528/    1265 : 1B 69                            DW Set_Recovery         ; cmnd 6
529/    1267 : 00 0C                            DW 0000Ch               ; cmnd 7 = Soft_Reset
530/    1269 : 1B 82                            DW Send_Park            ; cmnd 8
531/    126B : 19 99                            DW D_Read               ; cmnd 9
532/    126D : 19 B5                            DW D_ReadHdr            ; cmnd A
533/    126F : 1A 0B                            DW D_Write              ; cmnd B
534/    1271 : 1B 95                            DW Set_AutoOffset       ; cmnd C
535/    1273 : 18 67                            DW D_Read_SprTbl        ; cmnd D
536/    1275 : 1B A8                            DW Wr_SprTbl            ; cmnd E + password
537/    1277 : 1B D4                            DW Format               ; cmnd F + password
538/    1279 : 1C 48                            DW D_Init_SprTbl        ; cmnd 10 + password
539/    127B : 1C 15                            DW Read_Abort           ; cmnd 11
540/    127D : 1C 35                            DW D_RstSrvo            ; cmnd 12
541/    127F : 10 59                            DW Scan_Vector          ; cmnd 13
542/    1281 :
543/    1281 : 1C 74            Sys_Inst        DW Sys_Read
544/    1283 : 1D 48                            DW Sys_Write
545/    1285 : 19 3D                            DW Sys_WrV
546/    1287 :
547/    1287 :
548/    1287 :
549/    1287 :                  ;*****************************************************************
550/    1287 :                  ;
551/    1287 :                  ; Procedure: Pro_Read {ProFile read}
552/    1287 :                  ;
553/    1287 :                  ;  This procedure emulates a ProFile read operation.
554/    1287 :                  ;
555/    1287 :                  ; Inputs: none
556/    1287 :                  ;
557/    1287 :                  ; Outputs: (none)
558/    1287 :                  ;
559/    1287 :                  ; Local Variables: BlockStatus: BYTE {R4}
560/    1287 :                  ;                  BlockNumber: 3 BYTES {R12:14}
561/    1287 :                  ;
562/    1287 :                  ; Global Variables Changed: LogicalBlockNumber
563/    1287 :                  ;
564/    1287 :                  ; Algorithm:
565/    1287 :                  ;
566/    1287 :                  ;  Begin
567/    1287 :                  ;    ClearStatus
568/    1287 :                  ;    LogicalBlockNumber:=Command.LogicalBlockNumber
569/    1287 :                  ;    Ack_Read(Pro_Read_Response)
570/    1287 :                  ;    BlockType, BlockNumber:=Get_Type(ProFile)
```

```
571/    1287 :                     ;    Seek_Type:=Access {seek without Auto-Offset}
572/    1287 :                     ;    SearchStatus, Element.Ptr:=OverLappedSeek(Read, BlockType,
573/    1287 :                     ;                                              BlockNumber, 0)
574/    1287 :                     ;    If not(Read_Common)
575/    1287 :                     ;      Then Data_Ex_Handler(Read_Common.ErrorType)
576/    1287 :                     ;      Else
577/    1287 :                     ;        If BlkStat.SprCode=Bad_Block
578/    1287 :                     ;          Then
579/    1287 :                     ;            BlockMove(Buf2Array, RDummy)
580/    1287 :                     ;            Data_Ex_Handler(SpareBlock)
581/    1287 :                     ;    Move4(StatusArray, Status)
582/    1287 :                     ;    Clr_Bsy(StatusArray)
583/    1287 :                     ;  End
584/    1287 :                     ;
585/    1287 :                     ;*****************************************************************
586/    1287 :
587/    1287 : E6 0A 02            Pro_Read        ld Wrk_Io+10,#Read_Response
588/    128A : D6 03 02                            call Ack_Read
589/    128D :                     ;
590/    128D : D6 1E 5F                            call ClrNormStat
591/    1290 : 0C 01                               ld R0, #Pro_Log_Offset
592/    1292 : D6 06 4E                            call Ld_LgclBlk         ; LogicalBlockNumber := ...
593/    1295 : 8C 00                               ld R8, #ProFile
594/    1297 : 2C 21                               ld R2, #Get_Type /256
595/    1299 : 3C 9B                               ld R3, #Get_Type #256
596/    129B : D6 04 AB                            call Bank_Call
597/    129E : A6 E8 08                            cp R8, #SprTbl_Type     ; spare table requested?
598/    12A1 : 6D 18 70                            jp Z, Read_SprTbl
599/    12A4 : A6 E8 04                            cp R8, #ID_Type         ; ID block requested?
600/    12A7 : 6D 18 21                            jp Z, Read_ID
601/    12AA :                     ; get 'real' data block
602/    12AA : E6 57 80                            ld Seek_Type, #Access
603/    12AD : E6 58 02                            ld Data_Type, #User_Type
604/    12B0 : 56 56 D7                            and DiskStat, #0FFh-Wr_Op-Seq_CachSrch
605/    12B3 : 2C 1F                               ld R2, #OverLap /256
606/    12B5 : 3C 8D                               ld R3, #OverLap #256
607/    12B7 : D6 04 AB                            call Bank_Call
608/    12BA : D6 12 F2                            call Read_Common
609/    12BD : 6B 08               Pro_Rd_Exit     jr Z, ProRd_Err
610/    12BF : 76 5A 02                            tm BlkStat, #B_Block    ; check if block read was a bad block
611/    12C2 : 6B 06                               jr Z, Rd_Leave
612/    12C4 :                     ;
613/    12C4 : D6 12 DB                            call Pro_Rd_BB
614/    12C7 : D6 16 B1            ProRd_Err       call Data_Ex_Handler
615/    12CA :                     ;
616/    12CA : E6 0A 01            Rd_Leave        ld Wrk_Io+10, #Init_Response
617/    12CD : B0 0B                               clr Wrk_Io+11           ; Cmnd_Pending, IBsy:=false
618/    12CF : D6 12 E6            Rd_Leave2       call Ld_Stand_Stat
619/    12D2 : E6 0C 10            Rd_Leave1       ld Wrk_Io+12, #StatusArray /256
620/    12D5 : E6 0D 15                            ld Wrk_Io+13, #StatusArray #256
621/    12D8 : 8D 02 5F                            jp Clr_Bsy
622/    12DB :                     ;
623/    12DB :
624/    12DB : 2C 20               Pro_Rd_BB       LD R2, #RBuf_To_Buf2 /256
625/    12DD : 3C 99                               LD R3, #RBuf_To_Buf2 #256
626/    12DF : D6 04 AB                            call Bank_Call
627/    12E2 : D6 13 41                            call Rd_SprBlock
628/    12E5 : AF                                  ret
629/    12E6 :                     ;*****************************************************************
630/    12E6 : 2C 14               Ld_Stand_Stat   LD R2, #CStatus0 /256
631/    12E8 : 3C 95                               ld R3, #CStatus0 #256
632/    12EA : EC 10                               ld R14, #StatusArray /256
633/    12EC : FC 15                               ld R15, #StatusArray #256
634/    12EE : D6 1E 9A                            call Move4_B0
635/    12F1 : AF                                  ret
636/    12F2 :
637/    12F2 :
638/    12F2 :                     ;*****************************************************************
639/    12F2 :                     ;
640/    12F2 :                     ; Function: Read_Common
641/    12F2 :                     ;
642/    12F2 :                     ;  This function is responsible for all the common functions
643/    12F2 :                     ;  performed by all read commands.
644/    12F2 :                     ;
645/    12F2 :                     ; Inputs: none
646/    12F2 :                     ;
647/    12F2 :                     ; Outputs: Read_Common:     BOOLEAN {zero flag, true if data exception}
648/    12F2 :                     ;          ErrorCode.Error: BOOLEAN {R0/Bit 7}
649/    12F2 :                     ;          ErrorCode.Type:  7 BITS {R0/Bits 6:0}
650/    12F2 :                     ;
651/    12F2 :                     ; Local Variables: Retry:     BYTE {R5}
652/    12F2 :                     ;                  ErrorCode: BYTE {R4}
653/    12F2 :                     ;
654/    12F2 :                     ; Algorithm:
655/    12F2 :                     ;
656/    12F2 :                     ;  Begin
657/    12F2 :                     ;    ErrorCode.Error:=false
658/    12F2 :                     ;    If not(ReadBlock)
659/    12F2 :                     ;      Then
660/    12F2 :                     ;        ErrorCode.Error:=true
661/    12F2 :                     ;        ErrorCode.Type:=Undetermined
662/    12F2 :                     ;        If Recovery
663/    12F2 :                     ;          Then
664/    12F2 :                     ;            If ServoError Or NoHeaderFound
665/    12F2 :                     ;              Then
666/    12F2 :                     ;                ErrorCode.Error:=false
667/    12F2 :                     ;                Retry:=4
668/    12F2 :                     ;                Repeat
669/    12F2 :                     ;                  ServoRecovery
670/    12F2 :                     ;                  Retry:=Retry-1
671/    12F2 :                     ;                Until not(Retry=0) Or ReadBlock
672/    12F2 :                     ;            If not(ReadBlock)
673/    12F2 :                     ;              Then
674/    12F2 :                     ;                ErrorCode.Error:=true
675/    12F2 :                     ;                ErrorCode.Type:=Undetermined
676/    12F2 :                     ;            If ServoError
677/    12F2 :                     ;              Then
678/    12F2 :                     ;                SetStatus(Byte0, Status_Servo_Error)
679/    12F2 :                     ;                Abort
680/    12F2 :                     ;            If NoHeaderFounf
681/    12F2 :                     ;              Then ErrorCode.Type:=BadBlock
682/    12F2 :                     ;              Else
683/    12F2 :                     ;               If ErrCnt>0
684/    12F2 :                     ;                 Then
685/    12F2 :                     ;                   If ReadErrorCount>2 And ReadErrorCount<10
686/    12F2 :                     ;                     Then ErrorCode.Type:=Spare.Block
687/    12F2 :                     ;                     Else
688/    12F2 :                     ;                       If ReadErrorCount=10
```

```
689/    12F2 :                      ;                         Then
690/    12F2 :                      ;                           If ECC
691/    12F2 :                      ;                             Then ErrorCode.Type:=SpareBlock
692/    12F2 :                      ;                             Else ErrorCode.Type:=BadBlock
693/    12F2 :                      ;                Else ErrorCode.Type:=ReadError
694/    12F2 :                      ;    ReadCommon:=not(ErrorCode.Error)
695/    12F2 :                      ;  End
696/    12F2 :                      ;
697/    12F2 :                      ;*****************************************************************
698/    12F2 :
699/    12F2 : 56 56 DF             Read_Common     and DiskStat, #0FFh-Wr_Op
700/    12F5 : B0 E4                                clr R4
701/    12F7 : D6 1E A9                             call ReadBlock
702/    12FA : EB 4F                                jr NZ, Rd_Cmn_End
703/    12FC : 4C 80                                ld R4, #RdError         ; ErrorCode:=Error, undetermined
704/    12FE : 76 24 80                             tm Excpt_Stat, #Recovery        ; If recovery Then ...
705/    1301 : 6B 48                                jr Z, Rd_Cmn_End
706/    1303 : 5C 04                                ld R5, #4               ; Retry:=4
707/    1305 : 76 26 50             Rd_Cmn_Lp       tm RdStat, #RdSrvoErr+RdNoHdrFnd
708/    1308 : 6B 28                                jr Z, Rd_Cmn_Crct
709/    130A :                      ;
710/    130A : 2C 29                                ld R2, #SrvoRcvry /256
711/    130C : 3C 54                                ld R3, #SrvoRcvry #256
712/    130E : D6 04 AB                             call Bank_Call
713/    1311 : D6 1E A9                             call ReadBlock
714/    1314 : B0 E4                                clr R4
715/    1316 : EB 33                                jr NZ, Rd_Cmn_End
716/    1318 : 5A EB                                djnz R5, Rd_Cmn_Lp
717/    131A :                      ;
718/    131A : 76 26 40             Rd_Cmn_Servo    tm RdStat, #RdSrvoErr
719/    131D : 6B 0A                                jr Z, Rd_Cmn_Hdr
720/    131F : B0 E0                                clr R0                  ; Byte 0
721/    1321 : 1C 02                                ld R1, #Stat_Srvo
722/    1323 : D6 04 03                             call SetStatus
723/    1326 : D6 05 1F                             call Abort
724/    1329 :                      ;
725/    1329 : 76 26 10             Rd_Cmn_Hdr      tm RdStat, #RdNoHdrFnd
726/    132C : 6B 04                                jr Z, Rd_Cmn_Crct
727/    132E : 4C 88                                ld R4, #Error+Ex_HdrBad
728/    1330 : 8B 19                                jr Rd_Cmn_End
729/    1332 :                      ;
730/    1332 : 08 27                Rd_Cmn_Crct     ld R0, RdErrCnt
731/    1334 : 56 E0 0F                             and R0, #0Fh            ; mask off unwanted status
732/    1337 : A6 E0 03                             cp R0, #SprThresh
733/    133A : 2B 09                                jr LE, Rd_Cmn_RdErr
734/    133C : A6 E0 0A                             cp R0, #10
735/    133F : 6B 08                                jr Z, Rd_BadBlock
736/    1341 : 4C 82                Rd_SprBlock     ld R4, #Error+Ex_SprBlock       ; block spared: >10 ECC errors
737/    1343 : 8B 06                                jr Rd_Cmn_End
738/    1345 :                      ;
739/    1345 : 4C 86                Rd_Cmn_RdErr    ld R4, #Error+Ex_ReadErr
740/    1347 : 8B 02                                jr Rd_Cmn_End
741/    1349 :                      ;
742/    1349 : 4C 84                Rd_BadBlock     ld R4, #Error+Ex_BadBlock       ; bad block: >10 CRC errors
743/    134B :                      ;
744/    134B : 08 E4                Rd_Cmn_End      ld R0, R4
745/    134D : 66 E0 80                             tcm R0, #RdError
746/    1350 : 8D 04 F8                             jp Bank_Ret
747/    1353 :
748/    1353 :
749/    1353 :
750/    1353 :                      ;*****************************************************************
751/    1353 :                      ; Module Spare.Assem
752/    1353 :                      ;
753/    1353 :                      ; This module contains all the routines that pertain
754/    1353 :                      ; to the management of the spare table.
755/    1353 :                      ;
756/    1353 :                      ;       PROCEDURE SprBlock(SpareType : BIT {R10/bit 7})
757/    1353 :                      ;       PROCEDURE SpareBlock(BlockIsSpare : BOOLEAN {R10/bit 4})
758/    1353 :                      ;
759/    1353 :                      ;*****************************************************************
760/    1353 :
761/    1353 :                      ;*****************************************************************
762/    1353 :                      ;
763/    1353 :                      ; Procedure: SprBlock {spare a block}
764/    1353 :                      ;
765/    1353 :                      ;  This procedure is responsible for relocating a logical
766/    1353 :                      ;  block (note that the block type can be USER or SPARETABLE)
767/    1353 :                      ;  from either the user data area or the spare area to
768/    1353 :                      ;  some location within the spare area.
769/    1353 :                      ;
770/    1353 :                      ;  This procedure is capable of gobbling up ALL available spare
771/    1353 :                      ;  block space and will ABORT if no space is available.
772/    1353 :                      ;
773/    1353 :                      ; Inputs: SpareType: BIT {R10/Bit 4}
774/    1353 :                      ;
775/    1353 :                      ; Outputs: none
776/    1353 :                      ;
777/    1353 :                      ; Algorithm:
778/    1353 :                      ;
779/    1353 :                      ;  Begin
780/    1353 :                      ;    If SpareType=Spare
781/    1353 :                      ;      Then
782/    1353 :                      ;        If BlkStat.SpareCode=BadBlock
783/    1353 :                      ;          Then
784/    1353 :                      ;            DeleteSpare
785/    1353 :                      ;            SpareCount(Dec_BadCnt)
786/    1353 :                      ;        BlockMove(WBuffer1, Buffer2)
787/    1353 :                      ;        If not(WrVer_Common)
788/    1353 :                      ;          Then SpareBlock(SpareType)
789/    1353 :                      ;      Else SpareBlock(BadBlockType)
790/    1353 :                      ;    UpDate_SprTbl
791/    1353 :                      ;    BlockMove(Buffer2, RBuffer1)
792/    1353 :                      ;  End
793/    1353 :                      ;
794/    1353 :                      ;*****************************************************************
795/    1353 :
796/    1353 : 76 EA 10             SprBlock        tm R10, #Spare          ; If SpareType=Spare
797/    1356 : 6B 44                                jr Z, SprBlk_1
798/    1358 :                      ;
799/    1358 : D6 18 12                             call Get_Spr_Code
800/    135B : A6 E0 02                             cp R0, #B_Block
801/    135E : EB 0C                                jr NZ, SprBlk_2
802/    1360 :                      ;
803/    1360 : D6 15 AB                             call DeleteSpare
804/    1363 : 0C 02                                ld R0, #Dec_BadCnt
805/    1365 : 2C 25                                ld R2, #SpareCount /256
806/    1367 : 3C 19                                ld R3, #SpareCount #256
```

```
807/    1369 : D6 04 AB                         call Bank_Call
808/    136C : D6 05 CA         SprBlk_2        call ReSeek
809/    136F : 4C 0A                            ld R4, #10              ; bang on this block for a while
810/    1371 : 5C 00                            ld R5, #0               ; init error count
811/    1373 : 2C 20            Spr_WrVer       ld R2, #Buf2_To_WrBuf /256
812/    1375 : 3C 70                            ld R3, #Buf2_To_WrBuf #256
813/    1377 : D6 04 AB                         call Bank_Call
814/    137A : E6 27 0A                         ld RdErrCnt, #10        ; assume failure
815/    137D : D6 19 64                         call WrVer_Common
816/    1380 : EB 13                            jr NZ, SprWrV_1
817/    1382 : 44 28 28                         or WrStat, WrStat       ; check for write error
818/    1385 : EB 15                            jr NZ, SprBlk_1
819/    1387 : 08 27                            ld R0, RdErrCnt
820/    1389 : 56 E0 0F                         and R0, #0Fh            ; mask off status info
821/    138C : 6B 0E                            jr Z, SprBlk_1          ; check for header failure
822/    138E : A6 E0 03                         cp R0, #SprThresh
823/    1391 : AB 09                            jr GT, SprBlk_1         ; spare the block if over threshold
824/    1393 : 02 50                            add R5, R0              ; bump cumulative error count
825/    1395 : 4A DC            SprWrV_1        djnz R4, Spr_WrVer
826/    1397 : A6 E5 03                         cp R5, #SprThresh       ; take a percentage of total reads
827/    139A : 2B 03                            jr LE, SprBlk_3
828/    139C :                  ;
829/    139C : D6 13 AE         SprBlk_1        call SpareBlock
830/    139F :                  ;
831/    139F : 2C 24            SprBlk_3        ld R2, #UpDate_SprTbl /256
832/    13A1 : 3C 3F                            ld R3, #UpDate_SprTbl #256
833/    13A3 : D6 04 AB                         Call Bank_Call
834/    13A6 : 2C 20                            ld R2, #Buf2_To_RBuf /256
835/    13A8 : 3C 5B                            ld R3, #Buf2_To_RBuf #256
836/    13AA : D6 04 AB                         Call Bank_Call
837/    13AD : AF                               ret
838/    13AE :
839/    13AE :
840/    13AE :                  ;*****************************************************************
841/    13AE :                  ;
842/    13AE :                  ; Procedure: SpareBlock
843/    13AE :                  ;
844/    13AE :                  ;  This procedure performs the actual sparing.
845/    13AE :                  ;
846/    13AE :                  ; Inputs: SpareType: BIT {R10/Bit 4}
847/    13AE :                  ;
848/    13AE :                  ; Outputs: none
849/    13AE :                  ;
850/    13AE :                  ; Local Variables: SparingASpare: BOOLEAN {R9}
851/    13AE :                  ;                  SpareLocation: BYTE {R4}
852/    13AE :                  ;                  Error:         BOOLEAN {R5}
853/    13AE :                  ;
854/    13AE :                  ; Algorithm:
855/    13AE :                  ;
856/    13AE :                  ;  Begin
857/    13AE :                  ;    If BlkStat.SpareCode=BadBlock
858/    13AE :                  ;      Then
859/    13AE :                  ;        AddSpare(GetNewSpare(Load_Logical), Load_Logical, BadBlock)
860/    13AE :                  ;        SpareCount(Inc_BadCnt)
861/    13AE :                  ;      Else
862/    13AE :                  ;        Repeat
863/    13AE :                  ;          If BlkStat.SpareCode=Spare
864/    13AE :                  ;            Then
865/    13AE :                  ;              Location:=SrchSpTabl(Load_Logical)
866/    13AE :                  ;              Ptr:=Get_Ptr(Location)
867/    13AE :                  ;              Ptr^.Useable:=false
868/    13AE :                  ;          SpareCount(Inc_SprCnt)
869/    13AE :                  ;          BlkStat.SpareCode:=SpareBlock
870/    13AE :                  ;          Location:=GetNewSpare(Load_Logical)
871/    13AE :                  ;          AddSpare(Location, Load_Logical, SpareBlock)
872/    13AE :                  ;          Seek_Type:=Access_Offset
873/    13AE :                  ;          Seek(Get_Cyl_H_S(MulR0_m(Location)))
874/    13AE :                  ;        Until WrVer_Common
875/    13AE :                  ;  End
876/    13AE :                  ;
877/    13AE :                  ;*****************************************************************
878/    13AE :
879/    13AE : 0C 01            SpareBlock      ld R0, #1               ; byte 1
880/    13B0 : 1C 04                            ld R1, #Stat_Spare
881/    13B2 : D6 04 03                         call SetStatus
882/    13B5 : 76 EA 10                         tm R10, #Spare
883/    13B8 : EB 20                            jr NZ, S_Blk_Rpt
884/    13BA :                  ;
885/    13BA : D6 18 12                         call Get_Spr_Code
886/    13BD : A6 E0 02                         cp R0, #B_Block
887/    13C0 : 6B 25                            jr Z, S_Blk_End
888/    13C2 :                  ;
889/    13C2 : D6 06 0D                         call Load_Logical
890/    13C5 : D6 14 FB                         call GetNewSpare
891/    13C8 : F8 E0                            ld R15, R0
892/    13CA : 8C 00                            ld R8, #BadBlock
893/    13CC : D6 15 68                         call AddSpare
894/    13CF : 0C 01                            ld R0, #Inc_BadCnt
895/    13D1 : 2C 25                            ld R2, #SpareCount /256
896/    13D3 : 3C 19                            ld R3, #SpareCount #256
897/    13D5 : D6 04 AB                         call Bank_Call
898/    13D8 : 8B 0D                            jr S_Blk_End
899/    13DA :                  ;
900/    13DA : D6 06 0D         S_Blk_Rpt       call Load_Logical
901/    13DD : D6 13 EA                         call Spr_Enter
902/    13E0 : EB 05                            jr NZ, S_Blk_End
903/    13E2 : A6 E0 86                         cp R0, #Error+Ex_ReadErr        ; check for sparing threshold
904/    13E5 : EB F3                            jr NZ, S_Blk_Rpt
905/    13E7 : 8D 04 F8         S_Blk_End       jp Bank_Ret
906/    13EA :                  ;
907/    13EA : D6 18 12         Spr_Enter       call Get_Spr_Code
908/    13ED : A6 E0 01                         cp R0, #S_Block
909/    13F0 : EB 22                            jr NZ, S_Blk_New
910/    13F2 : D6 10 97                         call ExtPush_Vector
911/    13F5 : D6 14 6E                         call SrchSpTabl
912/    13F8 : 70 E1                            push R1                 ; save possible ptr to element
913/    13FA : 70 FC                            push FLAGS
914/    13FC : D6 10 9F                         call ExtPop_Vector
915/    13FF : 50 FC                            pop FLAGS
916/    1401 : 50 E1                            pop R1
917/    1403 : EB 03                            jr NZ, S_Blk_Unuse
918/    1405 : D6 05 1F                         call Abort
919/    1408 :                  ;
920/    1408 : 08 E1            S_Blk_Unuse     ld R0, R1
921/    140A : D6 17 F8                         call Get_Ptr
922/    140D : 82 02                            lde R0, @RR2            ; element useable:=false
923/    140F : 56 E0 DF                         and R0, #0FFh-Useable
924/    1412 : 92 02                            lde @RR2, R0
```

```
 925/    1414 :                     ;
 926/    1414 : 0C 00               S_Blk_New       ld R0, #Inc_SprCnt
 927/    1416 : 2C 25                               ld R2, #SpareCount /256
 928/    1418 : 3C 19                               ld R3, #SpareCount #256
 929/    141A : D6 04 AB                            call Bank_Call
 930/    141D : 56 5A FC                            and BlkStat, #0FCh      ; mask out the SpareCode stuff
 931/    1420 : 46 5A 01                            or BlkStat, #S_Block
 932/    1423 : D6 14 FB                            call GetNewSpare
 933/    1426 : F8 E0                               ld R15, R0
 934/    1428 : 8C 10                               ld R8, #Spare
 935/    142A : D6 15 68                            call AddSpare
 936/    142D : E6 57 90                            ld Seek_Type, #Access_Offset
 937/    1430 : 08 EF                               ld R0, R15              ; get location back
 938/    1432 : 0E                                  inc R0                  ; count 1..76
 939/    1433 : 2C 25                               ld R2, #MulR0_m /256
 940/    1435 : 3C CB                               ld R3, #MulR0_m #256
 941/    1437 : D6 04 AB                            call Bank_Call
 942/    143A : 2C 20                               ld R2, #Get_Cyl_H_S /256
 943/    143C : 3C D4                               ld R3, #Get_Cyl_H_S #256
 944/    143E : D6 04 AB                            call Bank_Call
 945/    1441 : 08 EF                               ld R0, R15
 946/    1443 : 2C 25                               ld R2, #ReMap_Sector /256
 947/    1445 : 3C EF                               ld R3, #ReMap_Sector #256
 948/    1447 : D6 04 AB                            call Bank_Call
 949/    144A : F8 E0                               ld R15, R0
 950/    144C : D6 10 8D                            call Seek_Vector
 951/    144F : A6 58 02                            cp Data_Type, #User_type
 952/    1452 : 6B 0D                               jr Z, Spr_LdBuf2
 953/    1454 : 2C 23                               ld R2, #SprChkSum /256  ; calculate new checkbyte
 954/    1456 : 3C E3                               ld R3, #SprChkSum #256
 955/    1458 : D6 04 AB                            call Bank_Call
 956/    145B : 2C 20                               ld R2, #Spr_To_WrBuf /256
 957/    145D : 3C AC                               ld R3, #Spr_To_WrBuf #256
 958/    145F : 8B 04                               jr Spr_LdWrBuf
 959/    1461 : 2C 20               Spr_LdBuf2      ld R2, #Buf2_To_WrBuf /256
 960/    1463 : 3C 70                               ld R3, #Buf2_To_WrBuf #256
 961/    1465 : D6 04 AB            Spr_LdWrBuf     call Bank_Call
 962/    1468 : D6 19 64                            call WrVer_Common
 963/    146B : 8D 04 F8                            jp Bank_Ret
 964/    146E :
 965/    146E :
 966/    146E :
 967/    146E :                     ;*****************************************************************
 968/    146E :                     ; Module Spare1.Assem (a continuation of Spare)
 969/    146E :                     ;
 970/    146E :                     ;       FUNCTION SrchSpTabl(LogicalBlock : 3 BYTES {R12:14}
 971/    146E :                     ;               ElementType: BYTE {R15}) :
 972/    146E :                     ;               PhysicalBlock : 3 BYTES {R12:14}
 973/    146E :                     ;               Status : BYTE {R0}
 974/    146E :                     ;               ElementPtr : BYTE {R1}
 975/    146E :                     ;       FUNCTION GetNewSpare(BlockNumber : 3 BYTES {R12:14}) : BYTE {R0}
 976/    146E :                     ;       PROCEDURE AddSpare(BlockType : 3 BITS {R8/bits 3:1}
 977/    146E :                     ;               SpareType : BIT {R8/bit 4}
 978/    146E :                     ;               Location  : BYTE {R15}
 979/    146E :                     ;               LogicalBlock : 3 BYTES {R12:14})
 980/    146E :                     ;       PROCEDURE DeleteSpare(Location : BYTE {R15}
 981/    146E :                     ;               LogicalBlock : 3 BYTES {R12:14})
 982/    146E :                     ;
 983/    146E :                     ;*****************************************************************
 984/    146E :
 985/    146E :                     ;*****************************************************************
 986/    146E :                     ;
 987/    146E :                     ; Function: SrchSpTabl {search spare table}
 988/    146E :                     ;
 989/    146E :                     ;  This procedure is responsible for checking to see if the
 990/    146E :                     ;  block number that is passed into it is currently in the
 991/    146E :                     ;  spare table. If hte block is found to be in the spare table
 992/    146E :                     ;  then the physical block number of the spare block is passed
 993/    146E :                     ;  back to the caller, as well as the PTR to the location of
 994/    146E :                     ;  spared block's element within the spare table. In any case,
 995/    146E :                     ;  a byte of status is always passed back to the caller describing
 996/    146E :                     ;  the state of the logical block within the spare table.
 997/    146E :                     ;
 998/    146E :                     ; Inputs: LogicalBlockNumber: 3 BYTES {R12:14}
 999/    146E :                     ;         ElementType:        BYTE {R15}
1000/    146E :                     ;
1001/    146E :                     ; Outputs: SrchSpTabl:  BOOLEAN {zero flag set if not found in table}
1002/    146E :                     ;          PhysicalBlockNumber: 3 BYTES {R12:14}
1003/    146E :                     ;          Status:              BYTE {R0}
1004/    146E :                     ;          ElementPtr:          BYTE {R1}
1005/    146E :                     ;
1006/    146E :                     ; Local Variables: HeadPtr: BYTE {R0}
1007/    146E :                     ;                  Ptr:     BYTE{R0, offset; RR8, actual Ptr}
1008/    146E :                     ;                  Found:   BYTE {R7}
1009/    146E :                     ;
1010/    146E :                     ; Algorithm:
1011/    146E :                     ;
1012/    146E :                     ;  Begin
1013/    146E :                     ;    Case DiskCapacity Of
1014/    146E :                     ;      10MB: k:=256; m:=256
1015/    146E :                     ;      20MB: k:=128; m:=512
1016/    146E :                     ;      40MB: k:= 64; m:=1024
1017/    146E :                     ;    HeadPtr:=Get_HeadPtr
1018/    146E :                     ;    If HeadPtr.Nil
1019/    146E :                     ;      Then PhysicalBlockNumber:=LogicalBlockNumber +
1020/    146E :                     ;                                LogicalBlockNumber DIV k
1021/    146E :                     ;      Else
1022/    146E :                     ;        Ptr:=HeadPtr.Ptr
1023/    146E :                     ;        Ptr:=Ptr*4 {calc offset into spare table}
1024/    146E :                     ;        Done:=false
1025/    146E :                     ;        Found:=false
1026/    146E :                     ;        While not(Done) Do
1027/    146E :                     ;          If Ptr^.Used And Ptr^.Useable And
1028/    146E :                     ;             Ptr^.Type=Element And
1029/    146E :                     ;             Ptr^.Token=LogicalBlockNumber/bits 0:9
1030/    146E :                     ;           Then
1031/    146E :                     ;             PhysicalBlock:=Ptr^.Location * m
1032/    146E :                     ;             Done:=true
1033/    146E :                     ;             Found:=true
1034/    146E :                     ;           Else
1035/    146E :                     ;             Ptr:=Ptr^.Ptr * 4
1036/    146E :                     ;             If Ptr^.Nil
1037/    146E :                     ;               Then Done:=true
1038/    146E :                     ;    Status:=Ptr^.Status
1039/    146E :                     ;    ElementPtr:=Ptr
1040/    146E :                     ;    If not(Found) Then SrchSpTabl:=false
1041/    146E :                     ;  End
1042/    146E :                     ;
```

```
1043/    146E :                     ;*********************************************************************
1044/    146E :
1045/    146E : D6 17 86            SrchSpTabl      call Get_HeadPtr
1046/    1471 : 6B 5F                               jr Z, NotInTabl
1047/    1473 :                     ;
1048/    1473 : B0 E7                               clr R7                  ; Found:=false
1049/    1475 : 09 41               SrchLp          ld ScrReg1, R0          ; save current ptr
1050/    1477 : D6 17 F8                            call Get_Ptr
1051/    147A : 82 12                               lde R1, @RR2            ; get element status
1052/    147C : 19 40                               ld ScrReg0, R1          ; save element status
1053/    147E : 76 E1 40                            tm R1, #Used            ; If Used
1054/    1481 : 6B 3B                               jr Z, SrchLpElse
1055/    1483 : 76 E1 20                            tm R1, #Useable         ;  And Useable
1056/    1486 : 6B 36                               jr Z, SrchLpElse
1057/    1488 : 08 58                               ld R0, Data_Type
1058/    148A : 72 10                               tm R1, R0               ;  And Type is correct
1059/    148C : 6B 30                               jr Z, SrchLpElse
1060/    148E : A0 E2                               incw RR2
1061/    1490 : 82 12                               lde R1, @RR2
1062/    1492 : 56 E1 03                            and R1, #3              ;  And (Token =
1063/    1495 : 08 ED                               ld R0, R13              ;   LogicalBlockNumber / bits 0:9)
1064/    1497 : 56 E0 03                            and R0, #3
1065/    149A : A2 01                               cp R0, R1
1066/    149C : EB 20                               jr NZ, SrchLpElse
1067/    149E : A0 E2                               incw RR2                ; point to bits 0:7 of token
1068/    14A0 : 82 02                               lde R0, @RR2
1069/    14A2 : A2 0E                               cp R0, R14
1070/    14A4 : EB 18                               jr NZ, SrchLpElse
1071/    14A6 : 76 40 10                            tm ScrReg0, #Spare      ; check if BadBlock
1072/    14A9 : EB 05                               jr NZ, Srch_Spare
1073/    14AB : 46 E7 01                            or R7, #Found
1074/    14AE : 8B 24                               jr Srch_Sp1
1075/    14B0 :                     ;
1076/    14B0 : 08 41               Srch_Spare      ld R0, ScrReg1
1077/    14B2 : 0E                                  inc R0                  ; number of spare blocks 1..76
1078/    14B3 :                     ; ***** INLINE: MulR0_m *****
1079/    14B3 : B0 EE                               clr R14                 ; Result:= R0 * 256
1080/    14B5 : D8 E0                               ld R13, R0
1081/    14B7 : B0 EC                               clr R12
1082/    14B9 : =>FALSE                     if W_20MB || W_40MB
1083/    14B9 :                                     rlc R14                 ; Result := Result * 2
1084/    14B9 :                                     rlc R13
1085/    14B9 :                                     rlc R12
1086/    14B9 : [1082]                      endif
1087/    14B9 : =>FALSE                     if W_40MB
1088/    14B9 :                                     rlc R14                 ; Result := Result * 2
1089/    14B9 :                                     rlc R13
1090/    14B9 :                                     rlc R12
1091/    14B9 : [1087]                      endif
1092/    14B9 :                     ; ***************************
1093/    14B9 : 46 E7 01                            or R7, #Found
1094/    14BC : 8B 34                               jr SrchDone
1095/    14BE :                     ;
1096/    14BE : 76 40 80            SrchLpElse      tm ScrReg0, #Nil        ; test if element.Ptr=Nil
1097/    14C1 : EB 0F                               jr NZ, NotInTabl
1098/    14C3 :                     ;
1099/    14C3 : 08 41                               ld R0, ScrReg1          ; get address of current element
1100/    14C5 : D6 17 F8                            call Get_Ptr
1101/    14C8 : 06 E3 03                            add R3, #3              ; point to next Ptr
1102/    14CB : 16 E2 00                            adc R2, #0
1103/    14CE : 82 02                               lde R0, @RR2
1104/    14D0 : 8B A3                               jr SrchLp
1105/    14D2 :                     ;
1106/    14D2 : B0 E7               NotInTabl       clr R7                  ; return Not_Found status
1107/    14D4 : D9 4D               Srch_Sp1        ld ScrRegD, R13
1108/    14D6 :                     ; ***** INLINE: Div3_k *****
1109/    14D6 : 28 ED                               ld R2, R13              ; shift right 1 byte
1110/    14D8 : 18 EC                               ld R1, R12
1111/    14DA : B0 E0                               clr R0
1112/    14DC : =>FALSE                     if W_20MB
1113/    14DC :                                     ld R15, #1              ; shift left once for DIV 128
1114/    14DC : [1112]                      endif
1115/    14DC : =>FALSE                     if W_40MB
1116/    14DC :                                     ld R15, #2
1117/    14DC : [1115]                      endif
1118/    14DC : =>FALSE                     if W_20MB || W_40MB
1119/    14DC :                                     ld R3, R14              ; shift left 1 bit
1120/    14DC :                     Div3_k_Lp       rlc R3                  ; move R3 bit 7 into carry flag
1121/    14DC :                                     rlc R2                  ; then shift all 3 bytes
1122/    14DC :                                     rlc R1
1123/    14DC :                                     rlc R0
1124/    14DC :                                     djnz R15, Div3_k_Lp
1125/    14DC : [1118]                      endif
1126/    14DC :                     ; ***************************
1127/    14DC : 02 E2                               add R14, R2             ; PhysicalBlock:=LogicalBlock +
1128/    14DE : 12 D1                               adc R13, R1             ;               LogicalBlock DIV k
1129/    14E0 : 12 C0                               adc R12, R0
1130/    14E2 : 08 ED                               ld R0, R13              ; save rollover byte
1131/    14E4 : =>TRUE                      if W_10MB
1132/    14E4 : A4 4D E0                            cp R0, ScrRegD          ; check for rollover
1133/    14E7 : [1131]                      endif
1134/    14E7 : =>FALSE                     if W_20MB
1135/    14E7 :                                     and R0, #0FEh           ; mask off MOD 512 bits
1136/    14E7 :                                     and ScrRegD, #0FEh
1137/    14E7 :                                     cp R0, ScrRegD          ; check for rollover
1138/    14E7 : [1134]                      endif
1139/    14E7 : =>FALSE                     if W_40MB
1140/    14E7 :                                     and R0, #0FCh           ; mask off MOD 1024 bits
1141/    14E7 :                                     and ScrRegD, #0FCh
1142/    14E7 :                                     cp R0, ScrRegD          ; check for rollover
1143/    14E7 : [1139]                      endif
1144/    14E7 : 6B 09                               jr Z, SrchDone
1145/    14E9 : 06 EE 01                            add R14, #1             ; otherwise account for rollover
1146/    14EC : 16 ED 00                            adc R13, #0
1147/    14EF : 16 EC 00                            adc R12, #0
1148/    14F2 :                     ;
1149/    14F2 : 42 77               SrchDone        or R7, R7               ; set status
1150/    14F4 : 08 40                               ld R0, ScrReg0          ; return status
1151/    14F6 : 18 41                               ld R1, ScrReg1          ; return ptr to element
1152/    14F8 : 8D 04 F8                            jp Bank_Ret
1153/    14FB :
1154/    14FB :
1155/    14FB :                     ;*********************************************************************
1156/    14FB :                     ;
1157/    14FB :                     ; Function: GetNewSpare
1158/    14FB :                     ;
1159/    14FB :                     ;  This function accepts either a physical block number or
1160/    14FB :                     ;  a logical block number and returns a one byte index into
```

```
1161/    14FB :                     ;  the Spare Table's bit map describing the location of the
1162/    14FB :                     ;  block that can be used as a spare
1163/    14FB :                     ;
1164/    14FB :                     ; Inputs: BlockNumber: 3 BYTES {R12:14}
1165/    14FB :                     ;
1166/    14FB :                     ; Outputs: GetNewSpare BYTE {R0}
1167/    14FB :                     ;
1168/    14FB :                     ; Global Variables Used: SpareBitMap
1169/    14FB :                     ;
1170/    14FB :                     ; Local Variables: Bit:   ScrReg0
1171/    14FB :                     ;                  Temp1: ScrReg1
1172/    14FB :                     ;                  Temp2: ScrReg2
1173/    14FB :                     ;                  NoHis: ScrReg3/bit 7
1174/    14FB :                     ;                  NoLos: ScrReg3/bit 6
1175/    14FB :                     ;
1176/    14FB :                     ; Algorithm:
1177/    14FB :                     ;
1178/    14FB :                     ;  Begin
1179/    14FB :                     ;    Bit:=SrchSpTabl div k {get physical blocknumber divided by
1180/    14FB :                     ;                           the number of blocks between spares }
1181/    14FB :                     ;    If SpareBitMap[Bit]=0
1182/    14FB :                     ;      Then GetNewSpare:=Bit
1183/    14FB :                     ;      Else
1184/    14FB :                     ;        NoHis:=false
1185/    14FB :                     ;        NoLos:=false
1186/    14FB :                     ;        Temp1:=Bit
1187/    14FB :                     ;        While not(NoHis) and SpareBitMap[Temp1]=1 Do
1188/    14FB :                     ;          Temp1:=Temp1+1
1189/    14FB :                     ;          If Temp1>=76 Then NoHis:=True
1190/    14FB :                     ;        Temp2:=Bit
1191/    14FB :                     ;        While not(NoLos) and SpareBitMap[Temp2]=1 Do
1192/    14FB :                     ;          Temp2:=Temp2-1
1193/    14FB :                     ;          If Temp2<0 Then NoLos:=true
1194/    14FB :                     ;        If NoHis and NoLos
1195/    14FB :                     ;          Then Abort {spare table full}
1196/    14FB :                     ;          Else
1197/    14FB :                     ;            If NoHis
1198/    14FB :                     ;              Then GetNewSpare:=Temp2
1199/    14FB :                     ;              Else
1200/    14FB :                     ;                If Temp1-Bit > Bit-Temp2
1201/    14FB :                     ;                  Then GetNewSpare:=Temp2
1202/    14FB :                     ;                  Else GetNewSpare:=Temp1
1203/    14FB :                     ;  End
1204/    14FB :                     ;
1205/    14FB :                     ;*****************************************************************
1206/    14FB :
1207/    14FB : D6 10 97            GetNewSpare     call ExtPush_Vector     ; save state
1208/    14FE : =>TRUE                      if W_10MB
1209/    14FE : 48 ED                       ld R4, R13              ; Div3_k, store result in R4
1210/    1500 : [1208]                      endif
1211/    1500 : D8 E4                       ld R13, R4
1212/    1502 : CC 80                       ld R12, #TestBitMap
1213/    1504 : D6 17 BE                    call TSC_BitMap         ; If BitMap[Bit]=0 ...
1214/    1507 : 08 E4                       ld R0, R4               ; assume Bit is unused
1215/    1509 : 6B 53                       jr Z, Gns_End           ; jump if Then
1216/    150B :                     ;
1217/    150B : 58 E4                       ld R5, R4               ; Else ...
1218/    150D : B0 E7                       clr R7
1219/    150F : D8 E5               Gns_Lp1         ld R13, R5              ; test for bit map location = 0
1220/    1511 : CC 80                       ld R12, #TestBitMap
1221/    1513 : D6 17 BE                    call TSC_BitMap
1222/    1516 : 6B 09                       jr Z, Gns_Lp1End
1223/    1518 : 5E                          inc R5                  ; bump Temp1
1224/    1519 : A6 E5 4C                    cp R5, #76
1225/    151C : 1B F1                       jr LT, Gns_Lp1
1226/    151E : 46 E7 80                    or R7, #80h             ; NoHis:=true
1227/    1521 : 68 E4               Gns_Lp1End      ld R6, R4
1228/    1523 : D8 E6               Gns_Lp2         ld R13, R6              ; test for bit map location = 0
1229/    1525 : CC 80                       ld R12, #TestBitMap
1230/    1527 : D6 17 BE                    call TSC_BitMap
1231/    152A : 6B 07                       jr Z, Gns_Lp2End
1232/    152C : 00 E6                       dec R6
1233/    152E : AB F3                       jr GT, Gns_Lp2
1234/    1530 : 46 E7 40                    or R7, #40h             ; NoLos:=true
1235/    1533 : 66 E7 C0            Gns_Lp2End      tcm R7, #0C0h           ; If NoHis and NoLos...
1236/    1536 : EB 0A                       jr NZ, Gns_Chk_Hi
1237/    1538 :                     ;
1238/    1538 : 0C 01                       ld R0, #1               ; status byte 1
1239/    153A : 1C 40                       ld R1, #SprBlk_Hard     ; spare table full
1240/    153C : D6 04 03                    call SetStatus
1241/    153F : D6 05 1F                    call Abort
1242/    1542 :                     ;
1243/    1542 : 08 E6               Gns_Chk_Hi      ld R0, R6               ; assume NoHis
1244/    1544 : 76 E7 80                    tm R7, #80h             ; test for NoHis
1245/    1547 : EB 15                       jr NZ, Gns_End
1246/    1549 :                     ;
1247/    1549 : 08 E5                       ld R0, R5               ; assume NoLos
1248/    154B : 76 E7 40                    tm R7, #40h             ; test for NoLos
1249/    154E : EB 0E                       jr NZ, Gns_End
1250/    1550 :                     ;
1251/    1550 : 18 E5                       ld R1, R5
1252/    1552 : 22 14                       sub R1, R4              ; otherwise find which is closer
1253/    1554 : 22 46                       sub R4, R6
1254/    1556 : 08 E6                       ld R0, R6               ; assume Temp2 is closer
1255/    1558 : A2 14                       cp R1, R4
1256/    155A : FB 02                       jr NC, Gns_End
1257/    155C : 08 E5                       ld R0, R5               ; otherwise Temp1 is closer
1258/    155E : 70 E0               Gns_End         push R0
1259/    1560 : D6 10 9F                    call ExtPop_Vector
1260/    1563 : 50 E0                       pop R0
1261/    1565 : 8D 04 F8                    jp Bank_Ret
1262/    1568 :
1263/    1568 :
1264/    1568 :                     ;*****************************************************************
1265/    1568 :                     ;
1266/    1568 :                     ; Procedure: AddSpare {add an element to the spare table}
1267/    1568 :                     ;
1268/    1568 :                     ;  This procedure is responsible for adding an element to the spare
1269/    1568 :                     ;  table. It accepts a 1 byte value describing the location within
1270/    1568 :                     ;  the spare table (it is assumed that the caller has already used
1271/    1568 :                     ;  GetNewSpare) as well as the LogicalBlockNumber and whether the
1272/    1568 :                     ;  block being added to the table is a Spare block or a Bad Block.
1273/    1568 :                     ;
1274/    1568 :                     ; Inputs: BlockType:          3 BITS {R8/bits 3:1}
1275/    1568 :                     ;         SpareType:          BOOLEAN {R8/bit4}
1276/    1568 :                     ;         Location:           BYTE {R15}
1277/    1568 :                     ;         LogicalBlockNumber: 3 BYTES {R12:14}
1278/    1568 :                     ;
```

```
1279/    1568 :                     ; Outputs: none
1280/    1568 :                     ;
1281/    1568 :                     ; Global Variables Used: SpareCount
1282/    1568 :                     ;
1283/    1568 :                     ; Algorithm:
1284/    1568 :                     ;
1285/    1568 :                     ;  Begin
1286/    1568 :                     ;    HeadPtr:=Get_HeadPtr(LogicalBlockNumber)
1287/    1568 :                     ;    If HeadPtr.Nil
1288/    1568 :                     ;      Then
1289/    1568 :                     ;        HeadPtr.Nil:=False
1290/    1568 :                     ;        HeadPtr.Ptr:=Location
1291/    1568 :                     ;        SegPtrArray[LogicalBlockNumber/bits 10:16]:= HeadPtr
1292/    1568 :                     ;      Else
1293/    1568 :                     ;        Ptr:=HeadPtr.Ptr
1294/    1568 :                     ;        Ptr:=Get_EoList(Ptr)
1295/    1568 :                     ;        Ptr^.Nil:=False
1296/    1568 :                     ;        Ptr^.Ptr:=Location
1297/    1568 :                     ;    Ptr:=Get_Ptr(Location)
1298/    1568 :                     ;    Ptr^.Nil:=True
1299/    1568 :                     ;    Ptr^.Used:=True
1300/    1568 :                     ;    Ptr^.Useable:=True
1301/    1568 :                     ;    Ptr^.Spare:=Spare
1302/    1568 :                     ;    Ptr^.Type:=SpareType
1303/    1568 :                     ;    Ptr^.Token:=LogicalBlockNumber/bits 0:9
1304/    1568 :                     ;    TCS_BitMap(Set, Location) {set the bit map location for the add}
1305/    1568 :                     ;    If SpareType=Spare
1306/    1568 :                     ;      Then SpareCount:=SpareCount+1
1307/    1568 :                     ;      Else BadCount:=BadCount+1
1308/    1568 :                     ;  End
1309/    1568 :                     ;
1310/    1568 :                     ;*********************************************************************
1311/    1568 :
1312/    1568 : D6 17 86            Addspare        call Get_HeadPtr        ; If HeadPtr.Nil ...
1313/    156B : EB 09                               jr NZ, ADS_Else1
1314/    156D :                     ;
1315/    156D : 56 E0 7F                            and R0, #0FFh-Nil       ; Then HeadPtr.Nil:=false
1316/    1570 : 42 0F                               or R0, R15              ;      HeadPtr.Ptr:=Location
1317/    1572 : 92 02                               lde @RR2, R0            ; create link
1318/    1574 : 8B 10                               jr ADS_UpDate
1319/    1576 :                     ;
1320/    1576 : D6 17 A3            ADS_Else1:      call Get_EoList         ; search till end of list
1321/    1579 : 56 E1 7F                            and R1, #0FFh-Nil       ; Ptr^.Nil:=false
1322/    157C : 92 12                               lde @RR2, R1            ; update table
1323/    157E : 06 E3 03                            add R3, #3              ; get Ptr^.Ptr
1324/    1581 : 16 E2 00                            adc R2, #0
1325/    1584 : 92 F2                               lde @RR2, R15           ; create link
1326/    1586 : 08 EF               ADS_UpDate      ld R0, R15              ; get structure ptr
1327/    1588 : D6 17 F8                            call Get_Ptr            ; create a read ptr out of it
1328/    158B : 0C E0                               ld R0, #Nil+Used+Useable
1329/    158D : 42 08                               or R0, R8               ; merge Spare/Bad Block/Type info
1330/    158F : 44 58 E0                            or R0, Data_Type
1331/    1592 : 92 02                               lde @RR2, R0
1332/    1594 : A0 E2                               incw RR2                ; point to element.HiToken
1333/    1596 : 08 ED                               ld R0, R13              ; get HiToken
1334/    1598 : 56 E0 03                            and R0, #3
1335/    159B : 92 02                               lde @RR2, R0
1336/    159D : A0 E2                               incw RR2                ; piont to element.LoToken
1337/    159F : 92 E2                               lde @RR2, R14           ; store LoToken
1338/    15A1 : CC 40                               ld R12, #SetBitMap
1339/    15A3 : D8 EF                               ld R13, R15
1340/    15A5 : D6 17 BE                            call TSC_BitMap         ; update the bit map
1341/    15A8 : 8D 04 F8                            jp Bank_Ret
1342/    15AB :
1343/    15AB :
1344/    15AB :                     ;*********************************************************************
1345/    15AB :                     ;
1346/    15AB :                     ; Procedure: DeleteSpare {delete an element from the spare table}
1347/    15AB :                     ;
1348/    15AB :                     ;  This procedure is responsible for deleting an element from the spare
1349/    15AB :                     ;  table. It accepts a 1 byte value describing the location within
1350/    15AB :                     ;  the spare table (it is assumed that the caller has already used
1351/    15AB :                     ;  GetNewSpare) as well as the LogicalBlockNumber.
1352/    15AB :                     ;
1353/    15AB :                     ; Inputs: Location:           BYTE {R15}
1354/    15AB :                     ;         LogicalBlockNumber: 3 BYTES {R12:14}
1355/    15AB :                     ;
1356/    15AB :                     ; Outputs: none
1357/    15AB :                     ;
1358/    15AB :                     ; Local Variables: Ptr1^.Status: BYTE {ScrRegF}
1359/    15AB :                     ;
1360/    15AB :                     ; Global Variables Used: SpareCount
1361/    15AB :                     ;
1362/    15AB :                     ; Algorithm:
1363/    15AB :                     ;
1364/    15AB :                     ;  Begin
1365/    15AB :                     ;    Location:=SrchSpTabl(Load_Logical)
1366/    15AB :                     ;    If not(SrchSpTabl.Found) Then Abort
1367/    15AB :                     ;    If Get_Head(LogicalBlockNumber).Ptr = Location
1368/    15AB :                     ;      Then Head.Nil:=true
1369/    15AB :                     ;      Else
1370/    15AB :                     ;        Ptr1:=Get_Ptr(Location)
1371/    15AB :                     ;        If Ptr1^.Nil=true
1372/    15AB :                     ;          Then Ptr(PreviousElement)^.Nil:=true
1373/    15AB :                     ;          Else Ptr(PreviousElement)^.Ptr:=Ptr1^.Ptr
1374/    15AB :                     ;    zero out the deleted element
1375/    15AB :                     ;    TCS_BitMap(Clear, Location) {free up the bit map location}
1376/    15AB :                     ;  End
1377/    15AB :                     ;
1378/    15AB :                     ;*********************************************************************
1379/    15AB :
1380/    15AB : D6 06 0D            DeleteSpare     call Load_Logical
1381/    15AE : D6 14 6E                            call SrchSpTabl
1382/    15B1 : F8 E1                               ld R15, R1
1383/    15B3 : EB 03                               jr NZ, Chk_HdPtr
1384/    15B5 : D6 05 1F                            call Abort
1385/    15B8 :                     ;
1386/    15B8 : D6 06 0D            Chk_HdPtr       call Load_Logical
1387/    15BB : D6 17 86                            call Get_HeadPtr        ; If Get_HeadPtr ...
1388/    15BE : 09 4E                               ld ScrRegE, R0
1389/    15C0 : 08 EF                               ld R0, R15
1390/    15C2 : D6 17 F8                            call Get_Ptr
1391/    15C5 : 82 02                               lde R0, @RR2
1392/    15C7 : 76 E0 80                            tm R0, #Nil
1393/    15CA : 6B 0E                               jr Z, Chk_Chain
1394/    15CC : A4 EF 4E                            cp ScrRegE, R15
1395/    15CF : EB 09                               jr NZ, Chk_Chain        ; Else ...
1396/    15D1 :                     ;
```

```
1397/    15D1 : D6 17 86                        call Get_HeadPtr
1398/    15D4 : 0C 80                           ld R0, #Nil             ; Then Head.Nil:=true
1399/    15D6 : 92 02                           lde @RR2, R0
1400/    15D8 : 8B 47                           jr Zero_Element
1401/    15DA :                    ;
1402/    15DA : 09 4F              Chk_Chain    ld ScrRegF, R0
1403/    15DC : 46 E0 80                        or R0, #Nil
1404/    15DF : 92 02                           lde @RR2, R0            ; break the chain
1405/    15E1 : 08 4E                           ld R0, ScrRegE
1406/    15E3 : D6 17 A3                        call Get_EoList         ; get Ptr(Previous) in ScrReg0,1
1407/    15E6 : 08 4F                           ld R0, ScrRegF          ; get the original status back
1408/    15E8 : 76 E0 80                        tm R0, #Nil             ; If Ptr1^.Nil
1409/    15EB : 28 40                           ld R2, ScrReg0          ; get Ptr(Previous)
1410/    15ED : 38 41                           ld R3, ScrReg1
1411/    15EF : 6B 09                           jr Z, D_Chk_Else        ; Else ...
1412/    15F1 :                    ;
1413/    15F1 : 82 02                           lde R0, @RR2
1414/    15F3 : 46 E0 80                        or R0, #80h
1415/    15F6 : 92 02                           lde @RR2, R0
1416/    15F8 : 8B 27                           jr Zero_Element
1417/    15FA :                    ;
1418/    15FA : A4 4E EF           D_Chk_Else   cp R15, ScrRegE
1419/    15FD : EB 05                           jr NZ, Get_Previous
1420/    15FF : D6 17 86                        call Get_HeadPtr
1421/    1602 : 8B 06                           jr Save_Previous
1422/    1604 :                    ;
1423/    1604 : 06 E3 03           Get_Previous add R3, #3              ; get to Ptr(Previous)^.Ptr
1424/    1607 : 16 E2 00                        adc R2, #0
1425/    160A : 70 E2              Save_Previous push R2
1426/    160C : 70 E3                           push R3
1427/    160E : 08 EF                           ld R0, R15
1428/    1610 : D6 17 F8                        call Get_Ptr
1429/    1613 : 06 E3 03                        add R3, #3              ; get to Ptr1^.Ptr
1430/    1616 : 16 E2 00                        adc R2, #0
1431/    1619 : 82 02                           lde R0, @RR2
1432/    161B : 50 E3                           pop R3
1433/    161D : 50 E2                           pop R2
1434/    161F : 92 02                           lde @RR2, R0            ; Ptr(Previous)^.Ptr:=Ptr1^.Ptr
1435/    1621 : 08 EF              Zero_Element ld R0, R15              ; get Ptr1 once more
1436/    1623 : D6 17 F8                        call Get_Ptr
1437/    1626 : 0C FF                           ld R0, #0FFh            ; initial element
1438/    1628 : 1C 04                           ld R1, #4               ; zero 4 bytes
1439/    162A : 92 02              Zero_E_Lp    lde @RR2, R0
1440/    162C : A0 E2                           incw RR2
1441/    162E : 1A FA                           djnz R1, Zero_E_Lp
1442/    1630 : CC 20                           ld R12, #ClearBitMap
1443/    1632 : D8 EF                           ld R13, R15
1444/    1634 : D6 17 BE                        call TSC_BitMap
1445/    1637 : 8D 04 F8                        jp Bank_Ret
1446/    163A :
1447/    163A :
1448/    163A :
1449/    163A :                    ;*********************************************************************
1450/    163A :                    ; Module Cache.Assem
1451/    163A :                    ;
1452/    163A :                    ; This name of this module is a bit misleading: there is no true
1453/    163A :                    ; cache implemented at this time. However, there is an attempt made
1454/    163A :                    ; at some primitive look-ahead methods to reduce the access time
1455/    163A :                    ; of the disk. This module contains those routines that are
1456/    163A :                    ; chiefly involved with the look-ahead algorithm.
1457/    163A :                    ;
1458/    163A :                    ;       FUNCTION CnvrtLogical(LogicalBlock : 3 BYTES {R12:14}) :
1459/    163A :                    ;               CnvrtLogical : BOOLEAN {true if block spared}
1460/    163A :                    ;               Cylinder : WORD {RR12}
1461/    163A :                    ;               Head : BYTE {R14}
1462/    163A :                    ;               Sector : BYTE {R15}
1463/    163A :                    ;               Status : BYTE {R0}
1464/    163A :                    ;               Ptr : BYTE {R1}
1465/    163A :                    ;       FUNCTION SrchCach(LogicalBlock : 3 BYTES {R12:14}
1466/    163A :                    ;               Random : BOOLEAN {R7/Bit 7}
1467/    163A :                    ;               Offset : 3 BITS {R7/Bits 2:0}) :
1468/    163A :                    ;               BOOLEAN
1469/    163A :                    ;               HeadSector : BYTE {R0}
1470/    163A :                    ;
1471/    163A :                    ;*********************************************************************
1472/    163A :
1473/    163A :                    ;*********************************************************************
1474/    163A :                    ;
1475/    163A :                    ; Function: CnvrtLogical  {convert logical block}
1476/    163A :                    ;
1477/    163A :                    ;  This function is responsible for converting a logical block
1478/    163A :                    ;  number to a physical block number by first searching the spare
1479/    163A :                    ;  table and then doing the appropriate arithmetic to arrive at
1480/    163A :                    ;  the cylinder, head, and sector value.
1481/    163A :                    ;
1482/    163A :                    ; Inputs: LogicalBlockNumber: 3 BYTES {R12:14}
1483/    163A :                    ;
1484/    163A :                    ; Outputs: Cylinder: WORD {R12:13}
1485/    163A :                    ;          Head:     BYTE {R14}
1486/    163A :                    ;          Sector:   BYTE {R15}
1487/    163A :                    ;          Status:   BYTE {R0, returned from search spare table}
1488/    163A :                    ;          Ptr:      BYTE {R1, returned from search spare table}
1489/    163A :                    ;
1490/    163A :                    ; Local Variables: PhysicalBlock: 3 BYTES {R12:14}
1491/    163A :                    ;                  Temp:          3 BYTES {R1:3}
1492/    163A :                    ;
1493/    163A :                    ; Algorithm:
1494/    163A :                    ;
1495/    163A :                    ;  Begin
1496/    163A :                    ;    If LogicalBlock>MaxLogicalBlock Then Abort
1497/    163A :                    ;    PhysicalBlock:=SearchSpareTable(LogicalBlock)
1498/    163A :                    ;    Cylinder:=PhysicalBlock div (Heads*Sectors)
1499/    163A :                    ;    Temp:=PhysicalBlock mod (Heads*Sectors)
1500/    163A :                    ;    Head:=Temp div Sectors
1501/    163A :                    ;    Sector:=Temp mod Sectors
1502/    163A :                    ;  End
1503/    163A :                    ;
1504/    163A :                    ;*********************************************************************
1505/    163A :
1506/    163A :                    CnvrtLogical
1507/    163A : D6 14 6E           Cnvrtsrch    call SrchSpTabl         ; check if logical block is in spare table
1508/    163D : 70 FC                           push FLAGS              ; save results
1509/    163F : 70 E0                           push R0                 ; save status
1510/    1641 : 70 E1                           push R1                 ; save ptr
1511/    1643 : 2C 20                           ld R2, #Get_Cyl_H_S /256
1512/    1645 : 3C D4                           ld R3, #Get_Cyl_H_S #256
1513/    1647 : D6 04 AB                        call Bank_Call          ; get cylinder, head, and sector
1514/    164A : 08 EF                           ld R0, R15              ; pass physical sector
```

```
1515/    164C : 2C 25                           ld R2, #ReMap_Sector /256
1516/    164E : 3C EF                           ld R3, #ReMap_Sector #256
1517/    1650 : D6 04 AB                        call Bank_Call
1518/    1653 : F8 E0                           ld R15, R0              ; R15:=logical sector
1519/    1655 : 50 E1                           pop R1                  ; Element.Ptr
1520/    1657 : 50 E0                           pop R0                  ; Search Status
1521/    1659 : 50 FC                           pop FLAGS
1522/    165B : 8D 04 F8                        jp Bank_Ret
1523/    165E :
1524/    165E :
1525/    165E :                     ;*****************************************************************
1526/    165E :                     ;
1527/    165E :                     ; Function: SrchCache
1528/    165E :                     ;
1529/    165E :                     ;  This function searches the block cache and returns both a boolean
1530/    165E :                     ;  variable (indicating whether the block is in the cache and if
1531/    165E :                     ;  there is a seek associated with it), as well as a info as to
1532/    165E :                     ;  whether a head select may be needed. The cache may be searched
1533/    165E :                     ;  sequentially, or randomly (as is the case for multiblock commands).
1534/    165E :                     ;
1535/    165E :                     ; Inputs: LogicalBlock: 3 BYTES {R12:14}
1536/    165E :                     ;         Offset:       BYTE {R0}
1537/    165E :                     ;         Working register set must be Wrk_Sys.
1538/    165E :                     ;
1539/    165E :                     ; Outputs: SrchCache:  BOOLEAN {zero flag is set if block is not
1540/    165E :                     ;                               in the cache or if there is a seek
1541/    165E :                     ;                               needed to get to the block}
1542/    165E :                     ;          HeadSector: BYTE {R0}
1543/    165E :                     ;
1544/    165E :                     ; Algorithm:
1545/    165E :                     ;
1546/    165E :                     ;  Begin
1547/    165E :                     ;    i:=1
1548/    165E :                     ;    SrchCache:=false
1549/    165E :                     ;    If CacheArray[Offset].LogicalBlock = LogicalBlock
1550/    165E :                     ;      Then
1551/    165E :                     ;        Found:=true
1552/    165E :                     ;        CacheStat:=CacheStatus[Offset]
1553/    165E :                     ;        If not(CacheStat.Seek)
1554/    165E :                     ;          Then SrchCache:=true
1555/    165E :                     ;  End
1556/    165E :                     ;
1557/    165E :                     ;*****************************************************************
1558/    165E :
1559/    165E : 31 40               SrchCache       srp #Wrk_Scr
1560/    1660 :                             assume RP:Wrk_Scr
1561/    1660 : 4C 14                       ld R4, #CacheLength
1562/    1662 : AC 16               Srch_C_Lp       ld R10, #CacheArray /256
1563/    1664 : BC FC                       ld R11, #CacheArray #256
1564/    1666 : 08 5B                       ld R0, Cache_Index
1565/    1668 : 90 E0                       rl R0                   ; multiply by 4
1566/    166A : 90 E0                       rl R0
1567/    166C : 02 B0                       add R11, R0             ; index into array
1568/    166E : 16 EA 00                    adc R10, #0
1569/    1671 : 8C 40                       ld R8, #ScrReg0
1570/    1673 : 83 8A                       ldei @R8, @RR10         ; load CacheArray.Logical
1571/    1675 : 83 8A                       ldei @R8, @RR10
1572/    1677 : 83 8A                       ldei @R8, @RR10
1573/    1679 : B4 1C E0                    xor R0, Wrk_Sys+12      ; compare values
1574/    167C : B4 1D E1                    xor R1, Wrk_Sys+13
1575/    167F : B4 1E E2                    xor R2, Wrk_Sys+14
1576/    1682 : 42 01                       or R0, R1               ; test for zero
1577/    1684 : 42 02                       or R0, R2
1578/    1686 : EB 17                       jr NZ, Srch_More
1579/    1688 :                     ;
1580/    1688 : 8C 16                       ld R8, #CacheStat /256  ; get CacheStatus
1581/    168A : 9C E8                       ld R9, #CacheStat #256
1582/    168C : 04 5B E9                    add R9, Cache_Index
1583/    168F : 20 5B                       inc Cache_Index
1584/    1691 : 82 18                       lde R1, @RR8
1585/    1693 : 19 5A                       ld BlkStat, R1          ; save search result
1586/    1695 : 82 0A                       lde R0, @RR10           ; get head/sector info
1587/    1697 : 31 10               Srch_C_End      srp #Wrk_Sys            ; get back to normal system context
1588/    1699 :                             assume RP:Wrk_Sys
1589/    1699 : 08 40                       ld R0, ScrReg0          ; pass head/sector info back
1590/    169B : 76 5A 80                    tm BlkStat, #CachSeek   ; set seek needed flag
1591/    169E : AF                          ret
1592/    169F :
1593/    169F : 20 5B               Srch_More       inc Cache_Index
1594/    16A1 : A6 5B 14                    cp Cache_Index, #CacheLength    ; check for cache overflow
1595/    16A4 : 1B 02                       jr LT, Srch_M_1
1596/    16A6 :                     ;
1597/    16A6 : B0 5B                       clr Cache_Index
1598/    16A8 : 4A B8               Srch_M_1        djnz R4, Srch_C_Lp
1599/    16AA : E6 5A 80                    ld BlkStat, #CachSeek
1600/    16AD : B0 5B                       clr Cache_Index
1601/    16AF : 8B E6                       jr Srch_C_End
1602/    16B1 :
1603/    16B1 :
1604/    16B1 :
1605/    16B1 :                     ;*****************************************************************
1606/    16B1 :                     ; Module Data.X.Assem
1607/    16B1 :                     ;
1608/    16B1 :                     ; Data Exception Handling
1609/    16B1 :                     ;
1610/    16B1 :                     ;       PROCEDURE: Data_Ex_Handler(ErrorCode : 7 BITS {R0/Bits 6:0})
1611/    16B1 :                     ;
1612/    16B1 :                     ;*****************************************************************
1613/    16B1 :
1614/    16B1 :                     ;*****************************************************************
1615/    16B1 :                     ;
1616/    16B1 :                     ; Procedure: Data_Ex_Handler  {data exception handler}
1617/    16B1 :                     ;
1618/    16B1 :                     ;  This procedure is responsible for cleaning up after some sort
1619/    16B1 :                     ;  of data error (spareing, bad-blocking, etc). It is
1620/    16B1 :                     ;  assumed that if spareing is going to occur that correct data must
1621/    16B1 :                     ;  be residing in Buffer2.
1622/    16B1 :                     ;
1623/    16B1 :                     ; Inputs: ErrorCode: 7 BITS {R0/Bits 6:0}
1624/    16B1 :                     ;
1625/    16B1 :                     ; Outputs: none
1626/    16B1 :                     ;
1627/    16B1 :                     ; Algorithm:
1628/    16B1 :                     ;
1629/    16B1 :                     ;  Begin
1630/    16B1 :                     ;    If not(DiskStatus.Wr_Op) Then SetStatus(ReadError)
1631/    16B1 :                     ;    Case ErrorCode Of
1632/    16B1 :                     ;      0: SetStatus(OperationFailed)
```

```
1633/    16B1 :                     ;      2: Spare(Spare)
1634/    16B1 :                     ;      4: Spare(BadBlock)
1635/    16B1 :                     ;         SetStatus(OperationFailed)
1636/    16B1 :                     ;      6: SetStatus(ErrorCount)
1637/    16B1 :                     ;      8: SetStatus(NoHeaderFound)
1638/    16B1 :                     ;         Spare(Spare)
1639/    16B1 :                     ;      Otherwise Abort
1640/    16B1 :                     ;  End
1641/    16B1 :                     ;
1642/    16B1 :                     ;*****************************************************************
1643/    16B1 :
1644/    16B1 : 70 E0               Data_Ex_Handler push R0                 ; save error code
1645/    16B3 : D6 10 97                            call ExtPush_Vector     ; save state
1646/    16B6 : 76 56 20                            tm DiskStat, #Wr_Op
1647/    16B9 : 6B 09                               jr Z, Data_Ex_RdErr
1648/    16BB : 0C 03                               ld R0, #3               ; load status byte 3
1649/    16BD : 18 29                               ld R1, WrErrCnt
1650/    16BF : D6 04 03                            call SetStatus
1651/    16C2 : 8B 03                               jr Data_Ex_Case
1652/    16C4 : D6 05 95            Data_Ex_RdErr   call SS_ReadErr
1653/    16C7 : 2C 2A               Data_Ex_Case    ld R2, #Set_SeekNeeded /256     ; invalidate cache
1654/    16C9 : 3C 0E                               ld R3, #Set_SeekNeeded #256
1655/    16CB : D6 04 AB                            call Bank_Call          ; update cache on next Pro/Sys command
1656/    16CE : 50 E1                               pop R1                  ; get error code back
1657/    16D0 : 56 E1 7F                            and R1, #7Fh            ; mask off error flag
1658/    16D3 : 76 E1 01                            tm R1, #1               ; see if error code is odd
1659/    16D6 : EB 05                               jr NZ, Data_Ex_Abort
1660/    16D8 : A6 E1 0A                            cp R1, #Ex_Case_Max     ; bounds check
1661/    16DB : 2B 03                               jr LE, Data_Ex_Cmnd
1662/    16DD : D6 05 1F            Data_Ex_Abort   call Abort
1663/    16E0 : 2C 16               Data_Ex_Cmnd    ld R2, #Ex_Jp_Tbl /256
1664/    16E2 : 3C F1                               ld R3, #Ex_Jp_Tbl #256
1665/    16E4 : 02 31                               add R3, R1              ; add offset
1666/    16E6 : 16 E2 00                            adc R2, #0
1667/    16E9 : C2 E2                               ldc R14, @RR2           ; get routine to execute
1668/    16EB : A0 E2                               incw RR2
1669/    16ED : C2 F2                               ldc R15, @RR2
1670/    16EF : 30 EE                               jp @RR14                ; go to routine
1671/    16F1 :
1672/    16F1 : 16 FD               Ex_Jp_Tbl       DW X_Undeter
1673/    16F3 : 17 02                               DW X_Spare
1674/    16F5 : 17 09                               DW X_BadBlock
1675/    16F7 : 17 1D                               DW X_ReadErr
1676/    16F9 : 17 2D                               DW X_HdrBadBlock
1677/    16FB : 17 53                               DW X_HdrSpare
1678/    16FD :
1679/    16FD : D6 05 87            X_Undeter       call SS_OpFail
1680/    1700 : 8B 25                               jr Except_Return
1681/    1702 :                     ;
1682/    1702 : AC 10               X_Spare         ld R10, #Spare
1683/    1704 : D6 13 53            X_Spr_Call      call SprBlock
1684/    1707 : 8B 1E                               jr Except_Return
1685/    1709 :                     ;
1686/    1709 : 4C 02               X_BadBlock      ld R4, #2               ; retry 2 times
1687/    170B : 70 E4               X_BB_Lp         push R4
1688/    170D : D6 17 58                            call Bad_Recover
1689/    1710 : 50 E4                               pop R4
1690/    1712 : EB EE                               jr NZ, X_Spare
1691/    1714 : 4A F5                               djnz R4, X_BB_Lp
1692/    1716 : D6 05 87                            call SS_OpFail
1693/    1719 : AC 00                               ld R10, #BadBlock
1694/    171B : 8B E7                               jr X_Spr_Call
1695/    171D :                     ;
1696/    171D : D6 05 95            X_ReadErr       call SS_ReadErr
1697/    1720 : 2C 20                               ld R2, #Buf2_To_RBuf /256
1698/    1722 : 3C 5B                               ld R3, #Buf2_To_RBuf #256
1699/    1724 : D6 04 AB                            call Bank_Call
1700/    1727 :                     ;
1701/    1727 : D6 10 9F            Except_Return   call ExtPop_Vector
1702/    172A : 8D 04 F8                            jp Bank_Ret
1703/    172D :                     ;
1704/    172D : D6 17 58            X_HdrBadBlock   call Bad_Recover
1705/    1730 : EB D0                               jr NZ, X_Spare
1706/    1732 : D6 05 A2                            call SS_NoHdr
1707/    1735 : D6 11 24                            call LocateSector
1708/    1738 : 2C 20                               ld R2, #ReadHdr /256
1709/    173A : 3C 07                               ld R3, #ReadHdr #256
1710/    173C : D6 04 AB                            call Bank_Call
1711/    173F : 6B 09                               jr Z, X_Hdr_Crct
1712/    1741 : 2C 20                               ld R2, #RBuf_To_Buf2 /256
1713/    1743 : 3C 99                               ld R3, #RBuf_To_Buf2 #256
1714/    1745 : D6 04 AB                            call Bank_Call
1715/    1748 : 8B B8                               jr X_Spare
1716/    174A :                     ;
1717/    174A : 2C 2F               X_Hdr_Crct      ld R2, #Ecc /256
1718/    174C : 3C 06                               ld R3, #Ecc #256
1719/    174E : D6 04 AB                            call Bank_Call
1720/    1751 : 6B B6                               jr Z, X_BadBlock
1721/    1753 :                     ;
1722/    1753 : D6 05 A2            X_HdrSpare      call SS_NoHdr
1723/    1756 : 8B AA                               jr X_Spare
1724/    1758 :
1725/    1758 :
1726/    1758 :                     ;*****************************************************************
1727/    1758 :                     ;
1728/    1758 :                     ; Function: Bad_Recover
1729/    1758 :                     ;
1730/    1758 :                     ;  This function is called when a block can not be read, either the
1731/    1758 :                     ;  header can not be found or there is a hard data error. The purpose
1732/    1758 :                     ;  here is to use the auto offset capabilities of the servo controller
1733/    1758 :                     ;  and reread the block.
1734/    1758 :                     ;
1735/    1758 :                     ; Inputs: none
1736/    1758 :                     ;
1737/    1758 :                     ; Outputs: Bad_Recover: BOOLEAN {zero flag set if failure on retry}
1738/    1758 :                     ;
1739/    1758 :                     ;*****************************************************************
1740/    1758 :
1741/    1758 : 76 56 01            Bad_Recover     tm DiskStat, #Offset_On ; check for auto_offset
1742/    175B : EB 03                               jr NZ, B_Rcvr_Read
1743/    175D : D6 05 CA                            call ReSeek
1744/    1760 : D6 12 F2            B_Rcvr_Read     call Read_Common
1745/    1763 : EB 20                               jr NZ, Bad_Rcvr_Ret
1746/    1765 : A6 E0 86                            cp R0, #Error+Ex_ReadErr
1747/    1768 : 6B 10                               jr Z, Bad_Rc_Spr
1748/    176A : A6 E0 84                            cp R0, #Error+Ex_BadBlock
1749/    176D : 6B 0F                               jr Z, Bad_Rc_Ecc
1750/    176F : A6 E0 82                            cp R0, #Error+Ex_SprBlock
```

```
1751/    1772 : 6B 06                           jr Z, Bad_Rc_Spr
1752/    1774 : 0C 00                           ld R0, #0
1753/    1776 : 42 00              Bad_Rc_Set   or R0, R0
1754/    1778 : 8B 0B                           jr Bad_Rcvr_Ret
1755/    177A : 0C 01              Bad_Rc_Spr   ld R0, #1
1756/    177C : 8B F8                           jr Bad_Rc_Set
1757/    177E : 2C 2F              Bad_Rc_Ecc   ld R2, #Ecc /256
1758/    1780 : 3C 06                           ld R3, #Ecc #256
1759/    1782 : D6 04 AB                        call Bank_Call
1760/    1785 : AF                 Bad_Rcvr_Ret ret
1761/    1786 :
1762/    1786 :
1763/    1786 :                    ;*********************************************************************
1764/    1786 :                    ; Module SprUtils.Assem
1765/    1786 :                    ;
1766/    1786 :                    ; This module contains all the primitive utility procedures used by
1767/    1786 :                    ; the module: Spare
1768/    1786 :                    ;
1769/    1786 :                    ;       FUNCTION Get_HeadPtr(LogicalBlock : 3 BYTES {R12:14}) :
1770/    1786 :                    ;               BOOLEAN
1771/    1786 :                    ;               HeadPtr : BYTE {R0}
1772/    1786 :                    ;               ArrayPtr : PTR {RR2}
1773/    1786 :                    ;               FUNCTION Get_EoList(Start_Ptr : BYTE {R0}) :
1774/    1786 :                    ;               SparePtr : BYTE {R0}
1775/    1786 :                    ;               ElementStatus : BYTE {R1}
1776/    1786 :                    ;               Ptr : PTR {RR2}
1777/    1786 :                    ;               PreviousPtr : PTR {ScrReg0:1}
1778/    1786 :                    ;       FUNCTION TSC_BitMap(TestBit : BOOLEAN {R12/bit 7}
1779/    1786 :                    ;               SetBit : BOOLEAN {R12/bit 6}
1780/    1786 :                    ;               ClearBit : BOOLEAN {R12/bit 5}
1781/    1786 :                    ;               Location : BYTE {R13}) : BOOLEAN
1782/    1786 :                    ;       FUNCTION Get_Ptr(SparePtr : BYTE {R0}) : Ptr : PTR {R2}
1783/    1786 :                    ;       FUNCTION Get_Spr_Code : 2 BITS {R0/Bits 1:0}
1784/    1786 :                    ;
1785/    1786 :                    ;*********************************************************************
1786/    1786 :
1787/    1786 :                    ;*********************************************************************
1788/    1786 :                    ;
1789/    1786 :                    ; Function: Get_HeadPtr
1790/    1786 :                    ;
1791/    1786 :                    ;  This function searches the SegPtrArray (that array for the spare
1792/    1786 :                    ;  table that contains the various head ptrs for the spare table) and
1793/    1786 :                    ;  returns the HeadPtr that corresponds to the list whose elements all
1794/    1786 :                    ;  have the first 7 bits of the (passed in) LogicalBlockNumber.
1795/    1786 :                    ;
1796/    1786 :                    ; Inputs: LogicalBlockNumber : 3 BYTES {R12:14}
1797/    1786 :                    ;
1798/    1786 :                    ; Outputs: Get_HeadPtr : BOOLEAN {zero is set if HeadPtr.Nil is true}
1799/    1786 :                    ;          HeadPtr     : BYTE {R0}
1800/    1786 :                    ;          ArrayPtr    : Ptr {RR2} {ptr to location in array}
1801/    1786 :                    ;
1802/    1786 :                    ; Algorithm:
1803/    1786 :                    ;
1804/    1786 :                    ;  BEGIN
1805/    1786 :                    ;    HeadPtr :=  SegPtrArray[ LogicalBlockNumber/bits 10:16 ]
1806/    1786 :                    ;    Get_HeadPtr := NOT( HeadPtr.Nil )
1807/    1786 :                    ;  END
1808/    1786 :                    ;
1809/    1786 :                    ;*********************************************************************
1810/    1786 :
1811/    1786 : 18 ED              Get_HeadPtr  ld R1, R13             ; bits 8:15 of logical block #
1812/    1788 : 56 E1 FC                        and R1, #0FCh
1813/    178B : 08 EC                           ld R0, R12           ; bit 16 of logical block #
1814/    178D : C0 E0                           rrc R0               ; put bit 16 into carry flag
1815/    178F : C0 E1                           rrc R1               ; shift bits 10:16 into !r1
1816/    1791 : CF                              rcf                  ; arithmetic divide by 2
1817/    1792 : C0 E1                           rrc R1
1818/    1794 : 2C 14                           ld R2, #SegPtrArray /256
1819/    1796 : 3C C5                           ld R3, #SegPtrArray #256
1820/    1798 : 02 31                           add R3, R1           ; add in offset to array
1821/    179A : 16 E2 00                        adc R2, #0
1822/    179D : 82 02                           lde R0, @RR2         ; load HeadPtr
1823/    179F : 66 E0 80                        tcm R0, #Nil         ; test for Nil Ptr
1824/    17A2 : AF                              ret
1825/    17A3 :
1826/    17A3 :                    ;*********************************************************************
1827/    17A3 :                    ;
1828/    17A3 :                    ; Function: Get_EoList (Get End-Of-List)
1829/    17A3 :                    ;
1830/    17A3 :                    ;  This function takes a starting ptr as an input parameter and
1831/    17A3 :                    ;  follows the list specified by the ptr until the list terminates.
1832/    17A3 :                    ;  A pointer to the last element of the array is returned to the
1833/    17A3 :                    ;  caller.
1834/    17A3 :                    ;
1835/    17A3 :                    ; Inputs: Start_Ptr : BYTE {R0}
1836/    17A3 :                    ;
1837/    17A3 :                    ; Outputs: Ptr : PTR {RR2}
1838/    17A3 :                    ;          SparePtr : BYTE {R0}
1839/    17A3 :                    ;          Element.Status : BYTE {R1}
1840/    17A3 :                    ;          PreviousPtr : PTR {ScrReg0,1}
1841/    17A3 :                    ;
1842/    17A3 :                    ; Algorithm:
1843/    17A3 :                    ;
1844/    17A3 :                    ;  BEGIN
1845/    17A3 :                    ;    Ptr := Get_Ptr( Start_Ptr )
1846/    17A3 :                    ;    WHILE NOT( Get_Ptr( Temp )^.Nil ) DO
1847/    17A3 :                    ;      PreviousPtr := Get_Ptr( Temp )
1848/    17A3 :                    ;      Temp := PreviuosPtr^.Ptr
1849/    17A3 :                    ;      Ptr := Get_Ptr( Temp )
1850/    17A3 :                    ;      Element.Status := Ptr^.Status
1851/    17A3 :                    ;  END
1852/    17A3 :                    ;
1853/    17A3 :                    ;*********************************************************************
1854/    17A3 :
1855/    17A3 : 09 43              Get_EoList   ld ScrReg3, R0
1856/    17A5 : D6 17 F8                        call Get_Ptr         ; create ptr into Spare Table
1857/    17A8 : 82 12                           lde R1, @RR2         ; get element status
1858/    17AA : 76 E1 80                        tm R1, #Nil          ; IF Nil THEN Exit Loop
1859/    17AD : EB 0E                           jr NZ, Get_EL_Done
1860/    17AF : 29 40                           ld ScrReg0, R2       ; save Ptr to element i - 1
1861/    17B1 : 39 41                           ld ScrReg1, R3
1862/    17B3 : 06 E3 03                        add R3, #3           ; get ptr to next
1863/    17B6 : 16 E2 00                        adc R2, #0
1864/    17B9 : 82 02                           lde R0, @RR2
1865/    17BB : 8B E6                           jr Get_EoList
1866/    17BD : AF                 Get_EL_Done  ret
1867/    17BE :
1868/
```

```
1869/    17BE :                     ;*********************************************************************
1870/    17BE :                     ;
1871/    17BE :                     ; Function: TSC_BitMap  (Test, Set, Or Clear BitMap Location)
1872/    17BE :                     ;
1873/    17BE :                     ;  This function performs one of 3 functions: Set a bit location in
1874/    17BE :                     ;  the Spare Table bit map; Clear a location in the Spare Table bit
1875/    17BE :                     ;  map; or Test a location in the spare table bit map. The location
1876/    17BE :                     ;  to test, set, or clear is an integer value between 0 and 75.
1877/    17BE :                     ;
1878/    17BE :                     ; Inputs: TestBit : BOOLEAN {R12/bit 7}
1879/    17BE :                     ;         SetBit  : BOOLEAN {R12/bit 6}
1880/    17BE :                     ;         ClearBit : BOOLEAN {R12/bit 5}
1881/    17BE :                     ;         Location : BYTE {R13}
1882/    17BE :                     ;
1883/    17BE :                     ; Outputs: TSC_BitMap : BOOLEAN {zero is set if location is zero}
1884/    17BE :                     ;
1885/    17BE :                     ; Global Variables Changed: SpareBitMap
1886/    17BE :                     ;
1887/    17BE :                     ;*********************************************************************
1888/    17BE :
1889/    17BE : 08 ED               TSC_BitMap      ld R0, R13
1890/    17C0 : 3C 03                               ld R3, #3               ; right hand justify the byte index
1891/    17C2 : CF                  TSC_Lp1         rcf
1892/    17C3 : C0 E0                               rrc R0
1893/    17C5 : 3A FB                               djnz R3, TSC_Lp1
1894/    17C7 : 2C 15                               ld R2, #SpareBitMap /256
1895/    17C9 : 3C 47                               ld R3, #SpareBitMap #256
1896/    17CB : 02 30                               add R3, R0
1897/    17CD : 16 E2 00                            adc R2, #0              ; inc second byte of address if needed
1898/    17D0 : 82 12                               lde R1, @RR2            ; load in the byte from bit map
1899/    17D2 : 56 ED 07                            and R13, #7             ; mask off all but MOD 8 of original
1900/    17D5 : 0C 80                               ld R0, #TestBitMap      ; convert index to mask
1901/    17D7 : 6B 04                               jr Z, TSC_Map
1902/    17D9 : E0 E0               TSC_Lp2         rr R0
1903/    17DB : DA FC                               djnz R13, TSC_Lp2
1904/    17DD : 70 E0               TSC_Map         push R0                 ; save mask
1905/    17DF : 76 EC 80                            tm R12, #TestBitMap
1906/    17E2 : EB 0F                               jr NZ, TSC_End
1907/    17E4 : 76 EC 40                            tm R12, #SetBitMap
1908/    17E7 : EB 06                               jr NZ, TSC_Set
1909/    17E9 : 60 E0               TSC_Clear       com R0
1910/    17EB : 52 10                               and R1, R0              ; mask out bit
1911/    17ED : 8B 02                               jr TSC_SCEnd
1912/    17EF :                     ;
1913/    17EF : 42 10               TSC_Set         or R1, R0               ; merge in bit
1914/    17F1 : 92 12               TSC_SCEnd       lde @RR2, R1            ; update the bit map
1915/    17F3 : 50 E0               TSC_End         pop R0
1916/    17F5 : 72 10                               tm R1, R0               ; test the bit
1917/    17F7 : AF                                  ret
1918/    17F8 :
1919/    17F8 :                     ;*********************************************************************
1920/    17F8 :                     ;
1921/    17F8 :                     ; Function: Get_Ptr
1922/    17F8 :                     ;
1923/    17F8 :                     ;  This function accepts a Spare Table ptr structure and creates a
1924/    17F8 :                     ;  real ptr into the Spare Table array.
1925/    17F8 :                     ;
1926/    17F8 :                     ; Inputs: SparePtr : BYTE {R0}
1927/    17F8 :                     ;
1928/    17F8 :                     ; Outputs: SParePtr : BYTE {R0} {input is unchanged}
1929/    17F8 :                     ;          Ptr : PTR {RR2}
1930/    17F8 :                     ;
1931/    17F8 :                     ;*********************************************************************
1932/    17F8 :
1933/    17F8 : 2C 15               Get_Ptr         ld R2, #SpareTable /256
1934/    17FA : 3C 51                               ld R3, #SpareTable #256
1935/    17FC : 70 E0                               push R0                 ; save structure ptr
1936/    17FE : 18 E0                               ld R1, R0
1937/    1800 : B0 E0                               clr R0
1938/    1802 : 02 11                               add R1, R1              ;  RR0 := 4 * R1
1939/    1804 : 02 11                               add R1, R1
1940/    1806 : 16 E0 00                            adc R0, #0
1941/    1809 : 02 31                               add R3, R1              ; add offset into table
1942/    180B : 12 20                               adc R2, R0
1943/    180D : 50 E0                               pop R0                  ; get original structure ptr back
1944/    180F : 8D 04 F8                            jp Bank_Ret
1945/    1812 :
1946/    1812 :                     ;*********************************************************************
1947/    1812 :                     ;
1948/    1812 :                     ; Function: Get_Spr_Code
1949/    1812 :                     ;
1950/    1812 :                     ;  This function returns the SpareCode portion of BlkStat
1951/    1812 :                     ;
1952/    1812 :                     ; Inputs: {none}
1953/    1812 :                     ;
1954/    1812 :                     ; Outputs: Get_Spr_Code: 2 BITS {R0/Bits 1:0}
1955/    1812 :                     ;
1956/    1812 :                     ;*********************************************************************
1957/    1812 :
1958/    1812 : 08 5A               Get_Spr_Code    ld R0, BlkStat
1959/    1814 : 56 E0 03                            and R0, #S_Block+B_Block        ; mask off all but SpareCode portion
1960/    1817 : AF                                  ret
1961/    1818 :
1962/    1818 :
1963/    1818 :
1964/    1818 :                     ;*********************************************************************
1965/    1818 :                     ; Module Cmnd0.Assem   {Command Driver 0}
1966/    1818 :                     ;
1967/    1818 :                     ;  This module controls the way in which commands received by the
1968/    1818 :                     ;  Host are executed. It's main responsibility is to determine whether
1969/    1818 :                     ;  a command string is protected by a check byte (the original Profile
1970/    1818 :                     ;  commands are not) and then decode the command and pass control over
1971/    1818 :                     ;  to the correct driver routine to execute the command. All exception
1972/    1818 :                     ;  handling at this level is controlled by the individual command
1973/    1818 :                     ;  routines.
1974/    1818 :                     ;
1975/    1818 :                     ;       PROCEDURE Read_ID
1976/    1818 :                     ;       PROCEDURE Read_SprTbl
1977/    1818 :                     ;       PROCEDURE Pro_Write
1978/    1818 :                     ;       FUNCTION Wr_Common : BOOLEAN
1979/    1818 :                     ;       PROCEDURE Pro_WrVer
1980/    1818 :                     ;       FUNCTION WrVer_Common : BOOLEAN
1981/    1818 :                     ;
1982/    1818 :                     ;*********************************************************************
1983/    1818 :
1984/    1818 :                     ;*********************************************************************
1985/    1818 :                     ;
1986/    1818 :                     ; Procedure: Read_ID
```

```
1987/    1818 :                     ;
1988/    1818 :                     ;  This procedure is responsible for letting the host system
1989/    1818 :                     ;  know what type of device it is talking to
1990/    1818 :                     ;
1991/    1818 :                     ; Inputs: none
1992/    1818 :                     ;
1993/    1818 :                     ; Outputs: none
1994/    1818 :                     ;
1995/    1818 :                     ; Algorithm:
1996/    1818 :                     ;
1997/    1818 :                     ;  Begin
1998/    1818 :                     ;    ClrNormStat
1999/    1818 :                     ;    ZeroRdBuf
2000/    1818 :                     ;    ID.DeviceName:=DeviceName
2001/    1818 :                     ;    ID.DeviceNumber:=DeviceNumber
2002/    1818 :                     ;    ID.Revision:=RevisionNumber
2003/    1818 :                     ;    ID.Capacity:=Capacity
2004/    1818 :                     ;    ID.BlockSize:=BlockSize
2005/    1818 :                     ;    Move4(StatusArray, Status)
2006/    1818 :                     ;    Goto Rd_Leave
2007/    1818 :                     ;  End
2008/    1818 :                     ;
2009/    1818 :                     ;*****************************************************************
2010/    1818 :
2011/    1818 : D6 1E 5F            D_Read_ID       call ClrNormStat
2012/    181B : E6 0A 02                            ld Wrk_Io+10, #D_R_ID_Response
2013/    181E : D6 03 02                            call Ack_Read
2014/    1821 :                     ;
2015/    1821 : 2C 20               Read_ID         ld R2, #Zero_RdBuf /256
2016/    1823 : 3C B6                               ld R3, #Zero_RdBuf #256
2017/    1825 : D6 04 AB                            call Bank_Call
2018/    1828 : CC 10                               ld R12, #RBuffer1 /256
2019/    182A : DC 19                               ld R13, #RBuffer1 #256
2020/    182C : EC 10                               ld R14, #DeviceParams /256
2021/    182E : FC A7                               ld R15, #DeviceParams #256
2022/    1830 : 1C 1E                               ld R1, #Dev_Parm_Length
2023/    1832 : C2 0E               Read_ID_Lp      ldc R0, @RR14
2024/    1834 : 92 0C                               lde @RR12, R0
2025/    1836 : A0 EC                               incw RR12
2026/    1838 : A0 EE                               incw RR14
2027/    183A : 1A F6                               djnz R1, Read_ID_Lp
2028/    183C : B0 E0                               clr R0
2029/    183E : B0 E1                               clr R1
2030/    1840 : EC 15                               ld R14, #SprCount /256
2031/    1842 : FC 45                               ld R15, #SprCount #256
2032/    1844 : 82 2E                               lde R2, @RR14           ; get spare count
2033/    1846 : D6 18 5E                            call Movel_3
2034/    1849 : A0 EE                               incw RR14               ; point to bad block count
2035/    184B : 82 2E                               lde R2, @RR14
2036/    184D : D6 18 5E                            call Movel_3
2037/    1850 : 2C 14                               ld R2, #FmtOffset /256
2038/    1852 : 3C C3                               ld R3, #FmtOffset #256
2039/    1854 : E8 EC                               ld R14, R12
2040/    1856 : F8 ED                               ld R15, R13
2041/    1858 : D6 1E 9A                            call Move4_B0
2042/    185B : 8D 12 CA                            jp Rd_Leave
2043/    185E :                     ;
2044/    185E : 3C 03               Movel_3         ld R3, #3               ; move 3 bytes
2045/    1860 : 48 FD                               ld R4, RP
2046/    1862 : 93 4C               Movel_3_Lp      ldei @RR12, @R4
2047/    1864 : 3A FC                               djnz R3, Movel_3_Lp
2048/    1866 : AF                                  ret
2049/    1867 :
2050/    1867 :
2051/    1867 :                     ;*****************************************************************
2052/    1867 :                     ;
2053/    1867 :                     ; Procedure: Read_SprTbl
2054/    1867 :                     ;
2055/    1867 :                     ;  This procedure is responsible for passing the spare table on to
2056/    1867 :                     ;  the host. Note that this information is sent in it's raw form and
2057/    1867 :                     ;  that it is up to the host to correctly interpret it.
2058/    1867 :                     ;
2059/    1867 :                     ; Inputs: none
2060/    1867 :                     ;
2061/    1867 :                     ; Outputs: none
2062/    1867 :                     ;
2063/    1867 :                     ; Algorithm:
2064/    1867 :                     ;
2065/    1867 :                     ;  Begin
2066/    1867 :                     ;    RBuffer1:=SpareArray
2067/    1867 :                     ;    Move4(StatusArray, Status)
2068/    1867 :                     ;    Clr_Bsy(StatusArray)
2069/    1867 :                     ;  End
2070/    1867 :                     ;
2071/    1867 :                     ;*****************************************************************
2072/    1867 :
2073/    1867 : D6 1E 5F            D_Read_SprTbl   call ClrNormStat
2074/    186A : E6 0A 0F                            ld Wrk_Io+10, #D_R_Spr_Resp
2075/    186D : D6 03 02                            call Ack_Read
2076/    1870 : 2C 20               Read_SprTbl     ld R2, #Spr_To_RBuf /256
2077/    1872 : 3C 7F                               ld R3, #Spr_To_RBuf #256
2078/    1874 : D6 04 AB                            call Bank_Call
2079/    1877 : 8D 12 CA                            jp Rd_Leave
2080/    187A :
2081/    187A :
2082/    187A :                     ;*****************************************************************
2083/    187A :                     ;
2084/    187A :                     ; Procedure: Pro_Write
2085/    187A :                     ;
2086/    187A :                     ;  This procedure emulates the ProFile write command.
2087/    187A :                     ;
2088/    187A :                     ; Inputs: none
2089/    187A :                     ;
2090/    187A :                     ; Outputs: none
2091/    187A :                     ;
2092/    187A :                     ; Algorithm:
2093/    187A :                     ;
2094/    187A :                     ;  Begin
2095/    187A :                     ;    ClrNormStat
2096/    187A :                     ;    LoadLogicalBlock(Pro_Log_Offset)
2097/    187A :                     ;    GetType(Widget)  {check logical block bounds}
2098/    187A :                     ;    Seek_Type:=Access_Offset
2099/    187A :                     ;    OverLap(User_Type, Write_Op, Write_Response, BlockNumber, 0)
2100/    187A :                     ;    If BlkStat.SprCode=BadBlock
2101/    187A :                     ;      Then
2102/    187A :                     ;        BlockMove(Buf2Array, WBuffer1-1)
2103/    187A :                     ;        Data_Ex_Handler(SpareBlock)
2104/    187A :                     ;      Else
```

```
2105/    187A :                     ;        If not(Wr_Common)
2106/    187A :                     ;          Then Data_Ex_Handler(Wr_Common.ErrorCode)
2107/    187A :                     ;    Rd_Leave
2108/    187A :                     ;  End
2109/    187A :                     ;
2110/    187A :                     ;*****************************************************************
2111/    187A :
2112/    187A : E6 0A 03            Pro_Write       ld Wrk_Io+10, #Wr_Response
2113/    187D : D6 18 95                            call Pro_Wr_SetUp
2114/    1880 : 76 5A 02                            tm BlkStat, #B_Block    ; If BadBlock
2115/    1883 : 6B 05                               jr Z, ProWr_1
2116/    1885 : D6 18 C2            Pro_Wr_BB       call Wr_BBlock
2117/    1888 : 8B 08                               jr Wr_Leave
2118/    188A : D6 18 CF            ProWr_1         call Wr_Common
2119/    188D : EB 03               ProWr_Exit      jr NZ, Wr_Leave
2120/    188F : D6 16 B1                            call Data_Ex_Handler
2121/    1892 : 8D 12 CA            Wr_Leave        jp Rd_Leave
2122/    1895 :                     ;
2123/    1895 : D6 1E 5F            Pro_Wr_SetUp    call ClrNormStat
2124/    1898 : D6 02 ED                            call Get_Wr_Data
2125/    189B : 0C 01                               ld R0, #Pro_Log_Offset
2126/    189D : D6 06 4E                            call Ld_LgclBlk         ; LogicalBlockNumber:=...
2127/    18A0 : 8C 01                               ld R8, #Widget
2128/    18A2 : 2C 21                               ld R2, #Get_Type /256
2129/    18A4 : 3C 9B                               ld R3, #Get_Type #256
2130/    18A6 : D6 04 AB                            call Bank_Call          ; check block bounds
2131/    18A9 : E6 57 90                            ld Seek_Type, #Access_Offset
2132/    18AC : E6 58 02                            ld Data_Type, #User_Type
2133/    18AF : 46 56 20                            or DiskStat, #Wr_Op
2134/    18B2 : 2C 1F                               ld R2, #OverLap /256
2135/    18B4 : 3C 8D                               ld R3, #OverLap #256
2136/    18B6 : D6 04 AB                            call Bank_Call
2137/    18B9 : 76 56 01                            tm DiskStat, #Offset_On
2138/    18BC : EB 03                               jr NZ, P_W_S_End
2139/    18BE : D6 05 CA                            call ReSeek
2140/    18C1 : AF                  P_W_S_End       ret
2141/    18C2 :
2142/    18C2 :                     ;*****************************************************************
2143/    18C2 : 2C 20               Wr_BBlock       ld R2, #WrBuf_To_Buf2 /256
2144/    18C4 : 3C 65                               ld R3, #WrBuf_To_Buf2 #256
2145/    18C6 : D6 04 AB                            call Bank_Call
2146/    18C9 : 0C 82                               ld R0, #Error+Ex_SprBlock
2147/    18CB : D6 16 B1                            call Data_Ex_Handler
2148/    18CE : AF                                  ret
2149/    18CF :
2150/    18CF :
2151/    18CF :                     ;*****************************************************************
2152/    18CF :                     ;
2153/    18CF :                     ; Function: Wr_Common
2154/    18CF :                     ;
2155/    18CF :                     ;  This function handles all of the write specific details
2156/    18CF :                     ;  for all of Widget's write commands.
2157/    18CF :                     ;
2158/    18CF :                     ; Inputs: none
2159/    18CF :                     ;
2160/    18CF :                     ; Outputs: Wr_Common:       BOOLEAN {zero flag is set if data exception}
2161/    18CF :                     ;          ErrorCode.Error: BOOLEAN {R0/Bit 7}
2162/    18CF :                     ;          ErrorCode.Type:  7 BITS {R0/Bits 6:0}
2163/    18CF :                     ;
2164/    18CF :                     ; Local Variables: Retry:     BYTE {R5}
2165/    18CF :                     ;                  ErrorCode: BYTE {R4}
2166/    18CF :                     ;
2167/    18CF :                     ; Algorithm:
2168/    18CF :                     ;
2169/    18CF :                     ;  Begin
2170/    18CF :                     ;    ErrorCode.Error:=false
2171/    18CF :                     ;      If not(WriteBlock)
2172/    18CF :                     ;        Then
2173/    18CF :                     ;          ErrorCode.Error:=true
2174/    18CF :                     ;          ErrorCode.Type:=Undetermined
2175/    18CF :                     ;          If Recovery
2176/    18CF :                     ;            Then
2177/    18CF :                     ;              If ServoError Or NoHeaderFound
2178/    18CF :                     ;                Then
2179/    18CF :                     ;                  ErrorCode.Error:=false
2180/    18CF :                     ;                  Retry:=4
2181/    18CF :                     ;                  Repeat
2182/    18CF :                     ;                    ServoRecovery
2183/    18CF :                     ;                    Retry:=Retry-1
2184/    18CF :                     ;                  Until not(Retry=0) or WriteBlock
2185/    18CF :                     ;                If not(WriteBlock)
2186/    18CF :                     ;                  Then
2187/    18CF :                     ;                    ErrorCode.Error:=true
2188/    18CF :                     ;                    ErrorCode.Type:=Undetermined
2189/    18CF :                     ;                If ServoError
2190/    18CF :                     ;                  Then
2191/    18CF :                     ;                    SetStatus(Byte0, Status_ServoError)
2192/    18CF :                     ;                    Abort
2193/    18CF :                     ;                If NoHeaderFound
2194/    18CF :                     ;                  Then
2195/    18CF :                     ;                    MoveBlock(Buffer2, WriteBuffer)
2196/    18CF :                     ;                    ErrorCode.Type:=SpareBlock
2197/    18CF :                     ;    Write_Common:=not(ErrorCode.Error)
2198/    18CF :                     ;  End
2199/    18CF :                     ;
2200/    18CF :                     ;*****************************************************************
2201/    18CF :
2202/    18CF : E6 42 12            Wr_Common       ld ScrReg2, #WrBlkFence /256    ; check for host overflow
2203/    18D2 : E6 43 70                            ld ScrReg3, #WrBlkFence #256
2204/    18D5 : 2C 24                               ld R2, #Chk_PassWord /256
2205/    18D7 : 3C E5                               ld R3, #Chk_PassWord #256
2206/    18D9 : D6 04 AB                            call Bank_Call
2207/    18DC : EB 0A                               jr NZ, Wr_Cmn_Ok
2208/    18DE :                     ;
2209/    18DE : 0C 00                               ld R0, #0               ; status byte zero
2210/    18E0 : 1C 40                               ld R1, #WrBuf_OR        ; buffer overrun
2211/    18E2 : D6 04 03                            call SetStatus
2212/    18E5 : D6 05 1F                            call Abort
2213/    18E8 : 70 E4               Wr_Cmn_Ok       push R4
2214/    18EA : 46 56 20                            or DiskStat, #Wr_Op
2215/    18ED : B0 E4                               clr R4
2216/    18EF : D6 1D B8                            call WriteBlock
2217/    18F2 : EB 3F                               jr NZ, Wr_Cmd_End
2218/    18F4 :                     ;
2219/    18F4 : 2C 20                               ld R2, #WrBuf_To_Buf2 /256
2220/    18F6 : 3C 65                               ld R3, #WrBuf_To_Buf2 #256
2221/    18F8 : D6 04 AB                            call Bank_Call
2222/    18FB : 4C 80                               ld R4, #Error           ; ErrorCode:=Error, undetermined
```

```
2223/    18FD : 76 24 80                        tm Excpt_Stat, #Recovery        ; If Recovery Then...
2224/    1900 : 6B 31                           jr Z, Wr_Cmd_End
2225/    1902 :                         ;
2226/    1902 : 5C 04                           ld R5, #4               ; Retry := 4
2227/    1904 : 2C 29                 Wr_Cmn_Lp ld R2, #SrvoRcvry /256
2228/    1906 : 3C 54                           ld R3, #SrvoRcvry #256
2229/    1908 : D6 04 AB                        call Bank_Call
2230/    190B : 2C 20                           ld R2, #Buf2_To_WrBuf /256
2231/    190D : 3C 70                           ld R3, #Buf2_To_WrBuf #256
2232/    190F : D6 04 AB                        call Bank_Call
2233/    1912 : D6 1D B8                        call WriteBlock
2234/    1915 : B0 E4                           clr R4
2235/    1917 : EB 1A                           jr NZ, Wr_Cmd_End
2236/    1919 : 5A E9                           djnz R5, Wr_Cmn_Lp
2237/    191B : 4C 80                           ld R4, #Error
2238/    191D :                         ;
2239/    191D : 76 28 40              Wr_Cmn_Servo tm WrStat, #WrSrvoErr
2240/    1920 : 6B 0A                           jr Z, Wr_Cmn_Hdr
2241/    1922 : B0 E0                           clr R0                  ; status byte 0
2242/    1924 : 1C 02                           ld R1, #Stat_Srvo
2243/    1926 : D6 04 03                        call SetStatus
2244/    1929 : D6 05 1F                        call Abort
2245/    192C :                         ;
2246/    192C : 76 28 10              Wr_Cmn_Hdr tm WrStat, #WrNoHdrFnd
2247/    192F : 6B 02                           jr Z, Wr_Cmd_End
2248/    1931 : 4C 8A                           ld R4, #Error+Ex_HdrSpr
2249/    1933 :                         ;
2250/    1933 : 08 E4                 Wr_Cmd_End ld R0, R4
2251/    1935 : 50 E4                           pop R4                  ; retrieve register from stack
2252/    1937 : 66 E0 80                        tcm R0, #WrError
2253/    193A : 8D 04 F8                        jp Bank_Ret
2254/    193D :
2255/    193D :
2256/    193D :                         ;*****************************************************************
2257/    193D :                         ;
2258/    193D :                         ; Procedure: Pro_WrVer
2259/    193D :                         ;
2260/    193D :                         ;  This procedure emulates the ProFile write/verify command.
2261/    193D :                         ;
2262/    193D :                         ; Inputs: none
2263/    193D :                         ;
2264/    193D :                         ; Outputs: none
2265/    193D :                         ;
2266/    193D :                         ; Algorithm:
2267/    193D :                         ;
2268/    193D :                         ;  Begin
2269/    193D :                         ;    ClrNormStat
2270/    193D :                         ;    LoadLogicalBlock(Pro_Log_Offset)
2271/    193D :                         ;    GetType(Widget)  {check logical block bounds}
2272/    193D :                         ;    Seek_Type:=Access_Offset
2273/    193D :                         ;    OverLap(User_Type, Write_Op, WrVer_Response, BlockNumber, 0)
2274/    193D :                         ;    If BlkStat.SprCode=BadBlock
2275/    193D :                         ;      Then
2276/    193D :                         ;        BlockMove(Buf2Array, WBuffer1-1)
2277/    193D :                         ;        Data_Ex_Handler(SpareBlock)
2278/    193D :                         ;      Else
2279/    193D :                         ;        If not(WrVer_Common)
2280/    193D :                         ;          Then Data_Ex_Handler(WrVer_Common.ErrorCode)
2281/    193D :                         ;    Rd_Leave
2282/    193D :                         ;  End
2283/    193D :                         ;
2284/    193D :                         ;*****************************************************************
2285/    193D :
2286/    193D : E6 0A 24              Sys_WrV   ld Wrk_Io+10, #Sys_WrVer_Resp
2287/    1940 : 8B 00                           jr Pro_WrVer
2288/    1942 : E6 0A 04              Pro_WrVer ld Wrk_Io+10, #WrVer_Response
2289/    1945 : D6 18 95              Pro_WrV_2 call Pro_Wr_SetUp
2290/    1948 : 2C 20                           ld R2, #WrBuf_To_Buf2 /256
2291/    194A : 3C 65                           ld R3, #WrBuf_To_Buf2 #256
2292/    194C : D6 04 AB                        call Bank_Call
2293/    194F : 76 5A 02                        tm BlkStat, #B_Block    ; If BadBlock
2294/    1952 : 6B 05                           jr Z, Pro_WrV_1
2295/    1954 : D6 18 C2                        call Wr_BBlock
2296/    1957 : 8B 08                           jr WrVer_Leave
2297/    1959 : D6 19 64              Pro_WrV_1 call WrVer_Common
2298/    195C : EB 03                           jr NZ, WrVer_Leave
2299/    195E : D6 16 B1                        call Data_Ex_Handler
2300/    1961 : 8D 12 CA              WrVer_Leave jp Rd_Leave
2301/    1964 :
2302/    1964 :
2303/    1964 :                         ;*****************************************************************
2304/    1964 :                         ;
2305/    1964 :                         ; Function: WrVer_Common
2306/    1964 :                         ;
2307/    1964 :                         ;  This function has the responsibility of making certain (within
2308/    1964 :                         ;  reasonable limits) thant when a block is written to the disk that
2309/    1964 :                         ;  it can also be read back. WriteVerify comes in two flavors:
2310/    1964 :                         ;  Conservative and NotConservative. The attempt here is to make
2311/    1964 :                         ;  the WriteVerify routine attractive to end users as well as provide
2312/    1964 :                         ;  a means for the controller to have a high degree of confidence
2313/    1964 :                         ;  in writing some of it's more important blocks.
2314/    1964 :                         ;
2315/    1964 :                         ; Inputs: none
2316/    1964 :                         ;
2317/    1964 :                         ; Outputs: WrVer_Common: BOOLEAN {zero flag is set if data exception}
2318/    1964 :                         ;
2319/    1964 :                         ; Algorithm:
2320/    1964 :                         ;
2321/    1964 :                         ;  Begin
2322/    1964 :                         ;    If Wr_Common
2323/    1964 :                         ;      Then
2324/    1964 :                         ;        DiskStat.Wr_Op:=False
2325/    1964 :                         ;        Rd_Common
2326/    1964 :                         ;        If Rd_Common.ErrorType=Ex_HdrBad
2327/    1964 :                         ;          Then Rd_Common.ErrorType:=Ex_HdrSpr
2328/    1964 :                         ;          Else
2329/    1964 :                         ;            If Rd_Common.ErrorType=Ex_BadBlock
2330/    1964 :                         ;              Then Rd_Common.ErrorType:=Ex_SprBlock
2331/    1964 :                         ;  End
2332/    1964 :                         ;
2333/    1964 :                         ;*****************************************************************
2334/    1964 :
2335/    1964 : D6 10 97              WrVer_Common call ExtPush_Vector
2336/    1967 : 46 56 20                        or DiskStat, #Wr_Op
2337/    196A : D6 18 CF                        call Wr_Common
2338/    196D : 6B 1D                           jr Z, WrVer_Ret
2339/    196F :                         ;
2340/    196F : 56 56 DF                        and DiskStat, #0FFh-Wr_Op
```

```
2341/    1972 : D6 12 F2                          call Read_Common
2342/    1975 : EB 15                             jr NZ, WrVer_Ret
2343/    1977 :                          ;
2344/    1977 : 18 E0                             ld R1, R0
2345/    1979 : 56 E1 7F                          and R1, #7Fh            ; mask off error bit
2346/    197C : A6 E1 08                          cp R1, #Ex_HdrBad
2347/    197F : 2C 8A                             ld R2, #Error+Ex_HdrSpr
2348/    1981 : 6B 07                             jr Z, WrV_ChgEx
2349/    1983 : A6 E1 04                          cp R1, #Ex_BadBlock
2350/    1986 : 2C 82                             ld R2, #Error+Ex_SprBlock
2351/    1988 : EB 02                             jr NZ, WrVer_Ret
2352/    198A : 08 E2                    WrV_ChgEx       ld R0, R2
2353/    198C : 70 E0                    WrVer_Ret       push R0
2354/    198E : D6 10 9F                          call ExtPop_Vector
2355/    1991 : 50 E0                             pop R0
2356/    1993 : 66 E0 80                          tcm R0, #Error          ; set error condition
2357/    1996 : 8D 04 F8                          jp Bank_Ret
2358/    1999 :
2359/    1999 :
2360/    1999 :
2361/    1999 :                          ;*****************************************************************
2362/    1999 :                          ; Module Cmnd1.Assem  {continuation of the command processor module}
2363/    1999 :                          ;
2364/    1999 :                          ;       PROCEDURE D_Read
2365/    1999 :                          ;       PROCEDURE D_ReadHdr
2366/    1999 :                          ;       PROCEDURE D_Write
2367/    1999 :                          ;       PROCEDURE Read_CStatus
2368/    1999 :                          ;       PROCEDURE Read_SStatus
2369/    1999 :                          ;       PROCEDURE Send_ServoCmnd
2370/    1999 :                          ;       PROCEDURE Send_Seek
2371/    1999 :                          ;       PROCEDURE Send_Restore
2372/    1999 :                          ;       PROCEDURE Send_Park
2373/    1999 :                          ;       PROCEDURE Set_Ram_Addr
2374/    1999 :                          ;       PROCEDURE Wr_SprTbl
2375/    1999 :                          ;       PROCEDURE Format
2376/    1999 :                          ;       PROCEDURE Read_Abort
2377/    1999 :                          ;       PROCEDURE D_RstSrvo
2378/    1999 :                          ;       PROCEDURE D_InitSprTbl
2379/    1999 :                          ;
2380/    1999 :                          ;*****************************************************************
2381/    1999 :
2382/    1999 :                          ;*****************************************************************
2383/    1999 :                          ;
2384/    1999 :                          ; Procedure: D_Read
2385/    1999 :                          ;
2386/    1999 :                          ;  This procedure is used by the Host to invoke the primitive read
2387/    1999 :                          ;  function. The block that is read by this routine is the last seek
2388/    1999 :                          ;  address (cylinder, head, and sector). It is best to turn recovery
2389/    1999 :                          ;  OFF before using this command - as it is when using all diagnostic
2390/    1999 :                          ;  commands.
2391/    1999 :                          ;
2392/    1999 :                          ; Inputs: none
2393/    1999 :                          ;
2394/    1999 :                          ; Outputs: none
2395/    1999 :                          ;
2396/    1999 :                          ; Algorithm:
2397/    1999 :                          ;
2398/    1999 :                          ;  Begin
2399/    1999 :                          ;    ClrNormStat
2400/    1999 :                          ;    Ack_Read(D_Read_Response)
2401/    1999 :                          ;    If not(Read_Common)
2402/    1999 :                          ;      Then Data_Ex_Handler(Read_Common.ErrorCode)
2403/    1999 :                          ;    Rd_Leave
2404/    1999 :                          ;  End
2405/    1999 :                          ;
2406/    1999 :                          ;*****************************************************************
2407/    1999 :
2408/    1999 : D6 1E 5F                 D_Read          call ClrNormStat
2409/    199C : E6 0A 0B                          ld Wrk_Io+10, #D_Read_Response
2410/    199F : D6 03 02                          call Ack_Read
2411/    19A2 : E6 58 02                          ld Data_Type, #User_Type
2412/    19A5 : 56 56 DF                          and DiskStat, #0FFh-Wr_Op
2413/    19A8 : D6 12 F2                          call Read_Common
2414/    19AB : EB 05                             jr NZ, D_Read_End
2415/    19AD : 0C 80                             ld R0, #Error
2416/    19AF : D6 16 B1                          call Data_Ex_Handler
2417/    19B2 : 8D 12 CA                 D_Read_End      jp Rd_Leave
2418/    19B5 :
2419/    19B5 :
2420/    19B5 :                          ;*****************************************************************
2421/    19B5 :                          ;
2422/    19B5 :                          ; Procedure: D_ReadHdr
2423/    19B5 :                          ;
2424/    19B5 :                          ;  This diagnostic command is used to read the header (and whatever
2425/    19B5 :                          ;  the contents may be) of the block pointed to by the last seek
2426/    19B5 :                          ;  address. The host should be cautioned that the buffer that the
2427/    19B5 :                          ;  controller points it to at the completion of the read is longer
2428/    19B5 :                          ;  than a normal read buffer by the length of the header and the
2429/    19B5 :                          ;  data gap.
2430/    19B5 :                          ;
2431/    19B5 :                          ; Inputs: none
2432/    19B5 :                          ;
2433/    19B5 :                          ; Outputs: none
2434/    19B5 :                          ;
2435/    19B5 :                          ; Algorithm:
2436/    19B5 :                          ;
2437/    19B5 :                          ;  Begin
2438/    19B5 :                          ;    ClrNormStat
2439/    19B5 :                          ;    Ack_Read(D_RdHdr_Response)
2440/    19B5 :                          ;    Sector:=LdParam1.First
2441/    19B5 :                          ;    LocateSector
2442/    19B5 :                          ;    ReadHdr
2443/    19B5 :                          ;    If ReadHdr.ServoErr Then SetStatus(Byte0, ServoErr)
2444/    19B5 :                          ;    If ReadHdr.CrcErr Then SetStatus(Byte0, ReadError)
2445/    19B5 :                          ;    Move4(RdHdrStatusArray, CStatus0)
2446/    19B5 :                          ;    Clr_Bsy(RdHdrStatusArray)
2447/    19B5 :                          ;  End
2448/    19B5 :                          ;
2449/    19B5 :                          ;*****************************************************************
2450/    19B5 :
2451/    19B5 : D6 1E 5F                 D_ReadHdr       call ClrNormStat
2452/    19B8 : E6 0A 0C                          ld Wrk_Io+10, #D_RdHdr_Resp
2453/    19BB : D6 03 02                          call Ack_Read
2454/    19BE : D6 1A E3                          call Ld_Param1
2455/    19C1 : 09 55                             ld Sector, R0           ; load new sector
2456/    19C3 : 2C 11                             ld R2, #LocateSector /256
2457/    19C5 : 3C 24                             ld R3, #LocateSector #256
2458/    19C7 : D6 04 AB                          call Bank_Call
```

```
2459/    19CA : E6 58 02                         ld Data_Type, #User_Type
2460/    19CD : 56 56 DF                         and DiskStat, #0FFh-Wr_Op
2461/    19D0 : 2C 20                            ld R2, #ReadHdr /256
2462/    19D2 : 3C 07                            ld R3, #ReadHdr #256
2463/    19D4 : D6 04 AB                         call Bank_Call
2464/    19D7 : 70 E0                            push R0                 ; save status byte
2465/    19D9 : 76 E0 40                         tm R0, #RdHSrvoErr      ; If ReadHdr.ServoErr
2466/    19DC : 6B 07                            jr Z, D_RdH_Crc
2467/    19DE : B0 E0                            clr R0                  ; status byte 0
2468/    19E0 : 1C 02                            ld R1, #Stat_Srvo
2469/    19E2 : D6 04 03                         call SetStatus
2470/    19E5 :                          ;
2471/    19E5 : 50 E0                    D_RdH_Crc       pop R0
2472/    19E7 : 76 E0 08                         tm R0, #RdCrcErr
2473/    19EA : 6B 07                            jr Z, D_RdH_End
2474/    19EC : B0 E0                            clr R0
2475/    19EE : 1C 08                            ld R1, #Stat_Rd_Err
2476/    19F0 : D6 04 03                         call SetStatus
2477/    19F3 :                          ;
2478/    19F3 : 2C 14                    D_RdH_End       ld R2, #CStatus0 /256
2479/    19F5 : 3C 95                            ld R3, #CStatus0 #256
2480/    19F7 : EC 10                            ld R14, #RdH_Stat_Array /256
2481/    19F9 : FC 08                            ld R15, #RdH_Stat_Array #256
2482/    19FB : D6 1E 9A                         call Move4_B0
2483/    19FE : 31 00                            srp #Wrk_Io
2484/    1A00 :                                  assume RP:Wrk_Io
2485/    1A00 : AC 01                            ld R10, #Init_Response
2486/    1A02 : B0 EB                            clr R11
2487/    1A04 : CC 10                            ld R12, #RdH_Stat_Array /256
2488/    1A06 : DC 08                            ld R13, #RdH_Stat_Array #256
2489/    1A08 : 8D 02 5F                         jp Clr_Bsy
2490/    1A0B :
2491/    1A0B :
2492/    1A0B :                          ;*****************************************************************
2493/    1A0B :                          ;
2494/    1A0B :                          ; Procedure: D_Write
2495/    1A0B :                          ;
2496/    1A0B :                          ;  This procedure, like D_Read, allows the host to perform a single
2497/    1A0B :                          ;  write to the last seek address. KEEP IN MIND THAT THIS ROUTINE
2498/    1A0B :                          ;  WILL ALLOW YOU TO WRITE ANYWHERE ON THE DISK THAT YOU CARE TO
2499/    1A0B :                          ;  WRITE!!!!
2500/    1A0B :                          ;
2501/    1A0B :                          ; Inputs: none
2502/    1A0B :                          ;
2503/    1A0B :                          ; Outputs: none
2504/    1A0B :                          ;
2505/    1A0B :                          ; Algorithm:
2506/    1A0B :                          ;
2507/    1A0B :                          ;  Begin
2508/    1A0B :                          ;    ClrNormStat
2509/    1A0B :                          ;    Get_Wr_Data(D_Write_Response)
2510/    1A0B :                          ;    If not(Write_Common)
2511/    1A0B :                          ;      Then Data_Ex_Handler(Wr_Common.ErrorCode)
2512/    1A0B :                          ;    Rd_Leave
2513/    1A0B :                          ;  End
2514/    1A0B :                          ;
2515/    1A0B :                          ;*****************************************************************
2516/    1A0B :
2517/    1A0B :                                  assume RP:Wrk_Sys
2518/    1A0B : D6 1E 5F                 D_Write         call ClrNormStat
2519/    1A0E : E6 58 02                         ld Data_Type, #User_Type
2520/    1A11 : 46 56 20                         or DiskStat, #Wr_Op
2521/    1A14 : E6 0A 0D                         ld Wrk_Io+10, #D_Write_Resp
2522/    1A17 : D6 02 ED                         call Get_Wr_Data
2523/    1A1A : D6 18 CF                         call Wr_Common
2524/    1A1D : EB 05                            jr NZ, D_Write_End
2525/    1A1F : 0C 80                            ld R0, #Error
2526/    1A21 : D6 16 B1                         call Data_Ex_Handler
2527/    1A24 : 8D 12 CA                 D_Write_End     jp Rd_Leave
2528/    1A27 :
2529/    1A27 :
2530/    1A27 :                          ;*****************************************************************
2531/    1A27 :                          ;
2532/    1A27 :                          ; Procedure: Read_CStatus
2533/    1A27 :                          ;
2534/    1A27 :                          ;  This procedure allows the host to read any of the controllers
2535/    1A27 :                          ;  status'. These status' all come in 4 byte quantities:
2536/    1A27 :                          ;
2537/    1A27 :                          ;  Status0: Standard_Status {sent to host at the completion
2538/    1A27 :                          ;                              of a command - note that a request
2539/    1A27 :                          ;                              for status is NOT a command}
2540/    1A27 :                          ;  Status1: Last_Logical_Block {0, HiByte; MidByte, LoByte}
2541/    1A27 :                          ;  Status2: Last_Seek_Address {HiCylinder, LoCylinder, Head, Sector}
2542/    1A27 :                          ;  Status3: Current_Cylinder {HiCylinder, LoCylinder, 0, 0}
2543/    1A27 :                          ;  Status4: Internal_Stat {Excpt_Stat, DiskStat, BlkStat, 0}
2544/    1A27 :                          ;  Status5: Cntrl_Port {0, SlfTst_Result, Port2, Controller_Port}
2545/    1A27 :                          ;  Status6: Error_Status {RdStat, RdErrCnt, WrStat, WrErrCnt}
2546/    1A27 :                          ;  Status7: Last_Seek_Address {Lst_HiCyl, Lst_LoCyl, Lst_Head,
2547/    1A27 :                          ;                              Lst_Sector}
2548/    1A27 :                          ;
2549/    1A27 :                          ;  {all illegal status requests will return standard status}
2550/    1A27 :                          ;
2551/    1A27 :                          ;*****************************************************************
2552/    1A27 :
2553/    1A27 :                                  assume RP:Wrk_Sys
2554/    1A27 : E6 0A 03                 Read_CStatus    ld Wrk_Io+10, #Rd_Stat_Resp
2555/    1A2A : D6 03 02                         call Ack_Read
2556/    1A2D : EC 10                            ld R14, #StatusArray /256
2557/    1A2F : FC 15                            ld R15, #StatusArray #256
2558/    1A31 : D6 1A E3                         call Ld_Param1
2559/    1A34 : 56 E0 0F                         and R0, #0Fh            ; mask off all but legal status request
2560/    1A37 : A6 E0 07                         cp R0, #7               ; check for except case
2561/    1A3A : AD 12 CA                         jp GT, Rd_Leave
2562/    1A3D :                          ;
2563/    1A3D : 2C 1A                            ld R2, #Stat_Table /256
2564/    1A3F : 3C 50                            ld R3, #Stat_Table #256
2565/    1A41 : 90 E0                            rl R0
2566/    1A43 : 02 30                            add R3, R0
2567/    1A45 : 16 E2 00                         adc R2, #0
2568/    1A48 : C2 C2                            ldc R12, @RR2
2569/    1A4A : A0 E2                            incw RR2
2570/    1A4C : C2 D2                            ldc R13, @RR2
2571/    1A4E : 30 EC                            jp @RR12
2572/    1A50 :
2573/    1A50 : 12 CA                    Stat_Table      DW Rd_Leave
2574/    1A52 : 1A 60                            DW Ld_Stat1
2575/    1A54 : 1A 6B                            DW Ld_Stat2
2576/    1A56 : 1A 75                            DW Ld_Stat3
```

```
2577/    1A58 : 1A 79                           DW Ld_Stat4
2578/    1A5A : 1A 83                           DW Ld_Stat5
2579/    1A5C : 1A 91                           DW Ld_Stat6
2580/    1A5E : 1A 9B                           DW Ld_Stat7
2581/    1A60 :
2582/    1A60 : A0 EE                 Ld_Stat1  incw RR14
2583/    1A62 : 2C 14                           ld R2, #LogicalBlock /256
2584/    1A64 : 3C A1                           ld R3, #LogicalBlock #256
2585/    1A66 : D6 1E 9A                        call Move4_B0
2586/    1A69 : 8B 40                           jr Ld_Stat_End
2587/    1A6B :                       ;
2588/    1A6B : 0C 52                 Ld_Stat2  ld R0, #Cylinder
2589/    1A6D : 1C 04                 Ld_Stat2_1 ld R1, #4                ; move 4 bytes
2590/    1A6F : 93 0E                 Ld_Stat2_Lp ldei @RR14, @R0
2591/    1A71 : 1A FC                           djnz R1, Ld_Stat2_Lp
2592/    1A73 : 8B 36                           jr Ld_Stat_End
2593/    1A75 :                       ;
2594/    1A75 : 0C 50                 Ld_Stat3  ld R0, #Cur_Cyl
2595/    1A77 : 8B F4                           jr Ld_Stat2_1
2596/    1A79 :                       ;
2597/    1A79 : 08 24                 Ld_Stat4  ld R0, Excpt_Stat
2598/    1A7B : 18 56                           ld R1, DiskStat
2599/    1A7D : 28 5A                           ld R2, BlkStat
2600/    1A7F : 38 58                           ld R3, Data_Type
2601/    1A81 : 8B 20                           jr Ld_Stat_Reg
2602/    1A83 :                       ;
2603/    1A83 : 0C 00                 Ld_Stat5  ld R0, #0
2604/    1A85 : 18 25                           ld R1, SlfTst_Result
2605/    1A87 : 28 02                           ld R2, P2                ; Port 2
2606/    1A89 : CC 1F                           ld R12, #StatusPort /256
2607/    1A8B : DC 00                           ld R13, #StatusPort #256
2608/    1A8D : 82 3C                           lde R3, @RR12
2609/    1A8F : 8B 12                           jr Ld_Stat_Reg
2610/    1A91 :                       ;
2611/    1A91 : 08 26                 Ld_Stat6  ld R0, RdStat
2612/    1A93 : 18 27                           ld R1, RdErrCnt
2613/    1A95 : 28 28                           ld R2, WrStat
2614/    1A97 : 38 29                           ld R3, WrErrCnt
2615/    1A99 : 8B 08                           jr Ld_Stat_Reg
2616/    1A9B :                       ;
2617/    1A9B : 08 5C                 Ld_Stat7  ld R0, Lst_HiCyl
2618/    1A9D : 18 5D                           ld R1, Lst_LoCyl
2619/    1A9F : 28 5E                           ld R2, Lst_Head
2620/    1AA1 : 38 5F                           ld R3, Lst_Sector
2621/    1AA3 :                       ;
2622/    1AA3 : 48 FD                 Ld_Stat_Reg ld R4, RP
2623/    1AA5 : 5C 04                           ld R5, #4                ; move 4 bytes
2624/    1AA7 : 93 4E                 Stat5_Lp  ldei @RR14, @R4
2625/    1AA9 : 5A FC                           djnz R5, Stat5_Lp
2626/    1AAB : E6 0A 01              Ld_Stat_End ld >Wrk_Io+10, #Init_Response
2627/    1AAE : B0 0B                           clr Wrk_Io+11
2628/    1AB0 : 8D 12 D2                        jp Rd_Leave1
2629/    1AB3 :
2630/    1AB3 :
2631/    1AB3 :                       ;*****************************************************************
2632/    1AB3 :                       ;
2633/    1AB3 :                       ; Procedure: Read_SStatus
2634/    1AB3 :                       ;
2635/    1AB3 :                       ;  This procedure is used by the host to examine the servo processor's
2636/    1AB3 :                       ;  status information. The controller makes NO assumptions concerning
2637/    1AB3 :                       ;  knowledge or meaning of the status bits, nor does the controller
2638/    1AB3 :                       ;  interpret the status requests sent by the host. In other words,
2639/    1AB3 :                       ;  when the host issues a request for status to the servo, it is
2640/    1AB3 :                       ;  his responsibility to make certain that this request is meaningful
2641/    1AB3 :                       ;
2642/    1AB3 :                       ;  The returned status is preceded by Standard Status.
2643/    1AB3 :                       ;
2644/    1AB3 :                       ;*****************************************************************
2645/    1AB3 :
2646/    1AB3 :                               assume RP:Wrk_Sys
2647/    1AB3 : D6 1E 5F              Read_SStatus  call ClrNormStat
2648/    1AB6 : E6 0A 04                        ld Wrk_Io+10, #Rd_SStat_Resp
2649/    1AB9 : D6 03 02                        call Ack_Read
2650/    1ABC : D6 1A E3                        call Ld_Param1
2651/    1ABF : D6 04 13                        call Set_Dmt
2652/    1AC2 : 09 43                           ld ScrReg3, R0
2653/    1AC4 : E6 40 00                        ld ScrReg0, #ReadStatus
2654/    1AC7 : E6 41 00                        ld ScrReg1, #0
2655/    1ACA : E6 42 00                        ld ScrReg2, #0
2656/    1ACD : 2C 28                           ld R2, #ServoStatus /256
2657/    1ACF : 3C 5D                           ld R3, #ServoStatus #256
2658/    1AD1 : D6 04 AB                        call Bank_Call
2659/    1AD4 : D6 04 1F                        call Clr_Dmt
2660/    1AD7 : D6 12 E6                        call Ld_Stand_Stat
2661/    1ADA : 2C 14                           ld R2, #SStatus0 /256
2662/    1ADC : 3C B6                           ld R3, #SStatus0 #256
2663/    1ADE : D6 1E 9A                        call Move4_B0
2664/    1AE1 : 8B C8                           jr Ld_Stat_End
2665/    1AE3 : E4 FD 44              Ld_Param1 ld ScrReg4, RP
2666/    1AE6 : 70 FD                           push RP
2667/    1AE8 : 31 40                           srp #Wrk_Scr
2668/    1AEA :                               assume RP:Wrk_Scr
2669/    1AEA : EC 14                           ld R14, #(CStatus4+2) /256
2670/    1AEC : FC A7                           ld R15, #(CStatus4+2) #256
2671/    1AEE : 5C 04                           ld R5, #4                ; load 4 bytes
2672/    1AF0 : 83 4E                 Rd_SStat_Lp ldei @R4, @RR14
2673/    1AF2 : 5A FC                           djnz R5, Rd_SStat_Lp
2674/    1AF4 : 50 FD                           pop RP                   ; get back to original context
2675/    1AF6 : AF                              ret
2676/    1AF7 :
2677/    1AF7 :
2678/    1AF7 :                       ;*****************************************************************
2679/    1AF7 :                       ;
2680/    1AF7 :                       ; Procedure: Send_ServoCmnd
2681/    1AF7 :                       ;
2682/    1AF7 :                       ;  This procedure is used by the host to send the servo a command
2683/    1AF7 :                       ;  of any type EXCEPT a status request (use Read_SStatus for that).
2684/    1AF7 :                       ;  The controller will Abort the operation if this servo command
2685/    1AF7 :                       ;  is sent to it.
2686/    1AF7 :                       ;
2687/    1AF7 :                       ;  Also, as in Read_SStatus, the host is responsible for the
2688/    1AF7 :                       ;  command string that it sends to the servo - all the Widget
2689/    1AF7 :                       ;  controller will do is pass along the command string.
2690/    1AF7 :                       ;
2691/    1AF7 :                       ;*****************************************************************
2692/    1AF7 :
2693/    1AF7 :                               assume RP:Wrk_Sys
2694/    1AF7 : D6 1E 5F              Send_ServoCmnd  call ClrNormStat
```

```
2695/    1AFA : E6 0A 05                        ld Wrk_Io+10, #Sd_S_C_Resp
2696/    1AFD : D6 03 02                        call Ack_Read
2697/    1B00 : D6 1A E3                        call Ld_Param1          ; get servo command
2698/    1B03 : 42 00                           or R0, R0               ; check for Status Command
2699/    1B05 : EB 03                           jr NZ, S_Scmnd
2700/    1B07 : D6 05 1F                        call Abort
2701/    1B0A : D6 04 13            S_Scmnd     call Set_Dmt
2702/    1B0D : 09 40                           ld ScrReg0, R0
2703/    1B0F : 19 41                           ld ScrReg1, R1
2704/    1B11 : 29 42                           ld ScrReg2, R2
2705/    1B13 : 39 43                           ld ScrReg3, R3
2706/    1B15 : 2C 28                           ld R2, #ServoCmnd /256
2707/    1B17 : 3C 37                           ld R3, #ServoCmnd #256
2708/    1B19 : D6 04 AB                        call Bank_Call
2709/    1B1C : D6 04 1F                        call Clr_Dmt
2710/    1B1F : 8D 12 CA                        jp Rd_Leave
2711/    1B22 :
2712/    1B22 :
2713/    1B22 :                     ;*****************************************************************
2714/    1B22 :                     ;
2715/    1B22 :                     ; Procedure: Send_Seek
2716/    1B22 :                     ;
2717/    1B22 :                     ;  This procedure allows the host to position the heads ANYWHERE
2718/    1B22 :                     ;  on the disk service that it pleases - the only consideration
2719/    1B22 :                     ;  is to be aware of the crash stop positions!
2720/    1B22 :                     ;
2721/    1B22 :                     ;  The form of the parameters passed into the controller are
2722/    1B22 :                     ;  HiCylinder, LowCylinder, Head, Sector. Keep in mind that these
2723/    1B22 :                     ;  values will be used to check for a valid header if and when
2724/    1B22 :                     ;  a state machine operation is initiated.
2725/    1B22 :                     ;
2726/    1B22 :                     ;*****************************************************************
2727/    1B22 :
2728/    1B22 : D6 1E 5F            Send_Seek       call ClrNormStat
2729/    1B25 : E6 0A 06                            ld Wrk_Io+10, #S_Seek_Response
2730/    1B28 : D6 03 02                            call Ack_Read
2731/    1B2B : D6 1A E3                            call Ld_Param1          ; get parameters from command processor
2732/    1B2E : E6 57 80                            ld Seek_Type, #Access
2733/    1B31 : C8 E0                               ld R12, R0              ; pass params to Seek
2734/    1B33 : D8 E1                               ld R13, R1
2735/    1B35 : E8 E2                               ld R14, R2
2736/    1B37 : F8 E3                               ld R15, R3
2737/    1B39 : D6 05 D7                            call New_Seek
2738/    1B3C : 2C 2A                               ld R2, #Set_SeekNeeded /256     ; invalidate cache
2739/    1B3E : 3C 0E                               ld R3, #Set_SeekNeeded #256
2740/    1B40 : D6 04 AB                            call Bank_Call
2741/    1B43 : 8D 12 CA                            jp Rd_Leave
2742/    1B46 :
2743/    1B46 :
2744/    1B46 :                     ;*****************************************************************
2745/    1B46 :                     ;
2746/    1B46 :                     ; Procedure: Send_Restore
2747/    1B46 :                     ;
2748/    1B46 :                     ;  This procedure allows the host to issue either a Format Recal or
2749/    1B46 :                     ;  a Data Recal to the servo without having to build a primitive
2750/    1B46 :                     ;  servo command. The value of which type of restore is to be
2751/    1B46 :                     ;  performed is passed in with the command string.
2752/    1B46 :                     ;
2753/    1B46 :                     ;*****************************************************************
2754/    1B46 :
2755/    1B46 : D6 1E 5F            Send_Restore    call ClrNormStat
2756/    1B49 : E6 0A 07                            ld Wrk_Io+10, #S_Rstr_Response
2757/    1B4C : D6 03 02                            call Ack_Read
2758/    1B4F : D6 1A E3                            call Ld_Param1          ; get recal type
2759/    1B52 : A6 E0 40                            cp R0, #DataRecal
2760/    1B55 : 6B 08                               jr Z, S_Restore
2761/    1B57 : A6 E0 70                            cp R0, #FrmtRecal
2762/    1B5A : 6B 03                               jr Z, S_Restore
2763/    1B5C : D6 05 1F                            call Abort
2764/    1B5F : 2C 28               S_Restore       ld R2, #Restore /256
2765/    1B61 : 3C E2                               ld R3, #Restore #256
2766/    1B63 : D6 04 AB                            call Bank_Call
2767/    1B66 : 8D 12 CA                            jp Rd_Leave
2768/    1B69 :
2769/    1B69 :
2770/    1B69 :                     ;*****************************************************************
2771/    1B69 :                     ;
2772/    1B69 :                     ; Procedure: Set_Recovery
2773/    1B69 :                     ;
2774/    1B69 :                     ;  This procedure is used by the host to set the global
2775/    1B69 :                     ;  variable Recovery either on or off. This variable controls
2776/    1B69 :                     ;  the behaviour of Widget when an exception occurs; in the
2777/    1B69 :                     ;  case of Recovery = true, the controller tries everything it
2778/    1B69 :                     ;  can to correct the exception and continue about it's day.
2779/    1B69 :                     ;  When Recovery = false, however, the controller will do nothing
2780/    1B69 :                     ;  about the problem and report the error to the host.
2781/    1B69 :                     ;
2782/    1B69 :                     ;*****************************************************************
2783/    1B69 :
2784/    1B69 : D6 1E 5F            Set_Recovery    call ClrNormStat
2785/    1B6C : E6 0A 08                            ld Wrk_Io+10, #Set_Rcvr_Resp
2786/    1B6F : D6 03 02                            call Ack_Read
2787/    1B72 : D6 1A E3                            call Ld_Param1                  ; get set/reset param
2788/    1B75 : 46 24 80                            or Excpt_Stat, #Recovery        ; assume true
2789/    1B78 : 42 00                               or R0, R0                       ; test for true or false
2790/    1B7A : EB 03                               jr NZ, Set_Rcvr_Store
2791/    1B7C : 56 24 7F                            and Excpt_Stat, #0FFh-Recovery  ; clear old bit
2792/    1B7F : 8D 12 CA            Set_Rcvr_Store  jp Rd_Leave
2793/    1B82 :
2794/    1B82 :
2795/    1B82 :                     ;*****************************************************************
2796/    1B82 :                     ;
2797/    1B82 :                     ; Procedure: Send_Park
2798/    1B82 :                     ;
2799/    1B82 :                     ;  This diagnostic command allows the host to issue a Park
2800/    1B82 :                     ;  command to the drive. This command has the effect of
2801/    1B82 :                     ;  sending the drive heads off the data surface.
2802/    1B82 :                     ;
2803/    1B82 :                     ; Inputs: none
2804/    1B82 :                     ;
2805/    1B82 :                     ; Outputs: none
2806/    1B82 :                     ;
2807/    1B82 :                     ; Algorithm:
2808/    1B82 :                     ;
2809/    1B82 :                     ;  Begin
2810/    1B82 :                     ;    ClrNormStat
2811/    1B82 :                     ;    Ack_Read(S_Park_Response)
2812/    1B82 :                     ;    Park_Heads
```

```
2813/    1B82 :                     ;    Rd_Leave
2814/    1B82 :                     ;  End
2815/    1B82 :                     ;
2816/    1B82 :                     ;*****************************************************************
2817/    1B82 :
2818/    1B82 : D6 1E 5F            Send_Park       call    ClrNormStat
2819/    1B85 : E6 0A 0A                            ld      Wrk_Io+10, #S_Park_Resp
2820/    1B88 : D6 03 02                            call    Ack_Read
2821/    1B8B : 2C 29                               ld      R2, #Park_Heads /256
2822/    1B8D : 3C 88                               ld      R3, #Park_Heads #256
2823/    1B8F : D6 04 AB                            call    Bank_Call
2824/    1B92 : 8D 12 CA                            jp      Rd_Leave
2825/    1B95 :
2826/    1B95 :
2827/    1B95 :                     ;*****************************************************************
2828/    1B95 :                     ;
2829/    1B95 :                     ; Procedure: Set_AutoOffset
2830/    1B95 :                     ;
2831/    1B95 :                     ;  This command allows the user to set the auto offset capabilities
2832/    1B95 :                     ;  of the drive without using the Servo Command function.
2833/    1B95 :                     ;
2834/    1B95 :                     ; Inputs: {none}
2835/    1B95 :                     ;
2836/    1B95 :                     ; Outputs: {none}
2837/    1B95 :                     ;
2838/    1B95 :                     ; Algorithm:
2839/    1B95 :                     ;
2840/    1B95 :                     ;  BEGIN
2841/    1B95 :                     ;    Auto_Offset
2842/    1B95 :                     ;  END
2843/    1B95 :                     ;
2844/    1B95 :                     ;*****************************************************************
2845/    1B95 :
2846/    1B95 : D6 1E 5F            Set_AutoOffset  call ClrNormStat
2847/    1B98 : E6 0A 0E                            ld Wrk_Io+10, #St_AO_Response
2848/    1B9B : D6 03 02                            call Ack_Read
2849/    1B9E : 2C 27                               ld R2, #Auto_Offset /256
2850/    1BA0 : 3C 9F                               ld R3, #Auto_Offset #256
2851/    1BA2 : D6 04 AB                            call Bank_Call
2852/    1BA5 : 8D 12 CA                            jp Rd_Leave
2853/    1BA8 :
2854/    1BA8 :
2855/    1BA8 :                     ;*****************************************************************
2856/    1BA8 :                     ;
2857/    1BA8 :                     ; Procedure: Wr_SprTbl
2858/    1BA8 :                     ;
2859/    1BA8 :                     ;  This command allows the host to update the spare table
2860/    1BA8 :                     ;  without the controller's intervention. BE CAREFUL WITH
2861/    1BA8 :                     ;  THIS ONE!
2862/    1BA8 :                     ;
2863/    1BA8 :                     ; Inputs: none
2864/    1BA8 :                     ;
2865/    1BA8 :                     ; Outputs: none
2866/    1BA8 :                     ;
2867/    1BA8 :                     ; Algorithm:
2868/    1BA8 :                     ;
2869/    1BA8 :                     ;  Begin
2870/    1BA8 :                     ;    If not(Chk_PassWord(CommandString#1)) Then Abort
2871/    1BA8 :                     ;    Ack_Read(Wr_Spr_Response)
2872/    1BA8 :                     ;    Clr_Bsy(SpareArray)
2873/    1BA8 :                     ;  End
2874/    1BA8 :                     ;
2875/    1BA8 :                     ;*****************************************************************
2876/    1BA8 :
2877/    1BA8 :                            assume RP:Wrk_Sys
2878/    1BA8 : E6 0A 10            Wr_SprTbl       ld Wrk_Io+10, #Wr_Spr_Resp
2879/    1BAB : D6 02 ED                            call Get_Wr_Data
2880/    1BAE : D6 1E 5F                            call ClrNormStat
2881/    1BB1 : E6 42 14                            ld ScrReg2, #(CStatus4+2) /256
2882/    1BB4 : E6 43 A7                            ld ScrReg3, #(CStatus4+2) #256
2883/    1BB7 : 2C 24                               ld R2, #Chk_PassWord /256
2884/    1BB9 : 3C E5                               ld R3, #Chk_PassWord #256
2885/    1BBB : D6 04 AB                            call Bank_Call
2886/    1BBE : EB 03                               jr NZ, Wr_Spr1
2887/    1BC0 : D6 05 1F                            call Abort
2888/    1BC3 :                     ;
2889/    1BC3 : 2C 20               Wr_Spr1         ld R2, #WrBuf_To_Spr /256
2890/    1BC5 : 3C 93                               ld R3, #WrBuf_To_Spr #256
2891/    1BC7 : D6 04 AB                            call Bank_Call
2892/    1BCA : 2C 24                               ld R2, #UpDate_SprTbl /256
2893/    1BCC : 3C 3F                               ld R3, #UpDate_SprTbl #256
2894/    1BCE : D6 04 AB                            call Bank_Call
2895/    1BD1 : 8D 12 CA                            jp Rd_Leave
2896/    1BD4 :
2897/    1BD4 :
2898/    1BD4 :                     ;*****************************************************************
2899/    1BD4 :                     ;
2900/    1BD4 :                     ; Procedure: Format
2901/    1BD4 :                     ;
2902/    1BD4 :                     ;  This procedure allows the host to format the track that the heads
2903/    1BD4 :                     ;  are currently positioned over.
2904/    1BD4 :                     ;
2905/    1BD4 :                     ;  BE ULTRA CAREFUL HERE!!!! THIS COMMAND DESTROYS ALL DATA ON THE
2906/    1BD4 :                     ;  TRACK.... DATA RECOVERY WILL BE IMPOSSIBLE AFTER A TRACK HAS BEEN
2907/    1BD4 :                     ;  FORMATTED.
2908/    1BD4 :                     ;
2909/    1BD4 :                     ;  The two parameters that are passed into the controller by the host
2910/    1BD4 :                     ;  allow the host to specify the offset from index mark (in sectors)
2911/    1BD4 :                     ;  that Sector 0 will be located and the interleave factor.
2912/    1BD4 :                     ;
2913/    1BD4 :                     ; Inputs: none
2914/    1BD4 :                     ;
2915/    1BD4 :                     ; Outputs: none
2916/    1BD4 :                     ;
2917/    1BD4 :                     ; Algorithm:
2918/    1BD4 :                     ;
2919/    1BD4 :                     ;  Begin
2920/    1BD4 :                     ;    ClrNormStat
2921/    1BD4 :                     ;    If not(Chk_PassWord(CStatus4+3)) Then Abort
2922/    1BD4 :                     ;    Ack_Read(Fmt_Response)
2923/    1BD4 :                     ;    If not(FormatTrack(CStatus4+1, CStatus4+2)
2924/    1BD4 :                     ;      Then Abort
2925/    1BD4 :                     ;    Rd_Leave
2926/    1BD4 :                     ;  End
2927/    1BD4 :                     ;
2928/    1BD4 :                     ;*****************************************************************
2929/    1BD4 :
2930/    1BD4 :                            assume RP:Wrk_Sys
```

```
2931/    1BD4 : E6 0A 11           Format          ld Wrk_Io+10, #Fmt_Response
2932/    1BD7 : D6 03 02                           call Ack_Read
2933/    1BDA : D6 1B EB                           call Chk_FmtParms
2934/    1BDD : 48 E0                              ld R4, R0
2935/    1BDF : 58 E1                              ld R5, R1
2936/    1BE1 : 2C 10                              ld R2, #FormatTrack /256
2937/    1BE3 : 3C CC                              ld R3, #FormatTrack #256
2938/    1BE5 : D6 04 AB                           call Bank_Call
2939/    1BE8 : 8D 12 CA                           jp Rd_Leave
2940/    1BEB :                    ;
2941/    1BEB : D6 1E 5F           Chk_FmtParms    call ClrNormStat
2942/    1BEE : E6 42 14                           ld ScrReg2, #(CStatus4+4) /256
2943/    1BF1 : E6 43 A9                           ld ScrReg3, #(CStatus4+4) #256
2944/    1BF4 : 2C 24                              ld R2, #Chk_PassWord /256
2945/    1BF6 : 3C E5                              ld R3, #Chk_PassWord #256
2946/    1BF8 : D6 04 AB                           call Bank_Call
2947/    1BFB : EB 03                              jr NZ, Format1
2948/    1BFD : D6 05 1F                           call Abort              ; password mismatch, bye!
2949/    1C00 :                    ;
2950/    1C00 : D6 1A E3           Format1         call Ld_Param1
2951/    1C03 : 76 E0 F0                           tm R0, #0FFh-(NbrSctrs-4)       ; check for illegal offset
2952/    1C06 : 6B 07                              jr Z, Fmt_Chk_Inter
2953/    1C08 : 98 E0              Format_Abort    ld R9, R0
2954/    1C0A : A8 E1                              ld R10, R1
2955/    1C0C : D6 05 1F                           call Abort
2956/    1C0F :                    ;
2957/    1C0F : A6 E1 06           Fmt_Chk_Inter   cp R1, #Max_InterLeave
2958/    1C12 : BB F4                              jr UGT, Format_Abort
2959/    1C14 : AF                                 ret
2960/    1C15 :
2961/    1C15 :
2962/    1C15 :                    ;*****************************************************************
2963/    1C15 :                    ;
2964/    1C15 :                    ; Procedure: Read_Abort
2965/    1C15 :                    ;
2966/    1C15 :                    ;  This command allows the host to get a snapshot of the controller's
2967/    1C15 :                    ;  register set a the time of the last abort. If there has been no
2968/    1C15 :                    ;  abort, the results of this command will be invalid.
2969/    1C15 :                    ;
2970/    1C15 :                    ; Inputs: none
2971/    1C15 :                    ;
2972/    1C15 :                    ; Outputs: none
2973/    1C15 :                    ;
2974/    1C15 :                    ;*****************************************************************
2975/    1C15 :
2976/    1C15 : D6 1E 5F           Read_Abort      call ClrNormStat
2977/    1C18 : E6 0A 13                           ld Wrk_Io+10, #Rd_Abrt_Resp
2978/    1C1B : D6 03 02                           call Ack_Read
2979/    1C1E : 0C 16                              ld R0, #Abort_Stat /256
2980/    1C20 : 1C D8                              ld R1, #Abort_Stat #256
2981/    1C22 : 2C 10                              ld R2, #RBuffer1 /256
2982/    1C24 : 3C 19                              ld R3, #RBuffer1 #256
2983/    1C26 :                    ; 1A44          ld R4, #128             ; copy 128 bytes
2984/    1C26 : 4C 00                              ld R4, #0               ; copy 256 bytes
2985/    1C28 : 82 50              Rd_Abt_Lp       lde R5, @RR0
2986/    1C2A : 92 52                              lde @RR2, R5
2987/    1C2C : A0 E0                              incw RR0
2988/    1C2E : A0 E2                              incw RR2
2989/    1C30 : 4A F6                              djnz R4, Rd_Abt_Lp
2990/    1C32 : 8D 12 CA                           jp Rd_Leave
2991/    1C35 :
2992/    1C35 :
2993/    1C35 :                    ;*****************************************************************
2994/    1C35 :                    ;
2995/    1C35 :                    ; Procedure: D_RstSrvo
2996/    1C35 :                    ;
2997/    1C35 :                    ;  This procedure allows the host to reset the servo processor.
2998/    1C35 :                    ;
2999/    1C35 :                    ; Inputs: none
3000/    1C35 :                    ;
3001/    1C35 :                    ; Outputs: none
3002/    1C35 :                    ;
3003/    1C35 :                    ;*****************************************************************
3004/    1C35 :
3005/    1C35 : D6 1E 5F           D_RstSrvo       call ClrNormStat
3006/    1C38 : E6 0A 14                           ld Wrk_Io+10, #RstSrvo_Resp
3007/    1C3B : D6 03 02                           call Ack_Read
3008/    1C3E : 2C 29                              ld R2, #ResetServo /256
3009/    1C40 : 3C A2                              ld R3, #ResetServo #256
3010/    1C42 : D6 04 AB                           call Bank_Call
3011/    1C45 : 8D 12 CA                           jp Rd_Leave
3012/    1C48 :
3013/    1C48 :
3014/    1C48 :                    ;*****************************************************************
3015/    1C48 :                    ;
3016/    1C48 :                    ; Procedure: D_Init_SprTbl
3017/    1C48 :                    ;
3018/    1C48 :                    ;  This command allows the host to initialize the spare table
3019/    1C48 :                    ;  of the disk.
3020/    1C48 :                    ;
3021/    1C48 :                    ; Inputs: none
3022/    1C48 :                    ;
3023/    1C48 :                    ; Outputs: none
3024/    1C48 :                    ;
3025/    1C48 :                    ;*****************************************************************
3026/    1C48 :
3027/    1C48 :                            assume RP:Wrk_Sys
3028/    1C48 : D6 1E 5F           D_Init_SprTbl   call ClrNormStat
3029/    1C4B : E6 0A 12                           ld Wrk_Io+10, #I_Spr_Response
3030/    1C4E : D6 03 02                           call Ack_Read
3031/    1C51 : E6 42 14                           ld ScrReg2, #(CStatus4+4) /256
3032/    1C54 : E6 43 A9                           ld ScrReg3, #(CStatus4+4) #256
3033/    1C57 : 2C 24                              ld R2, #Chk_PassWord /256
3034/    1C59 : 3C E5                              ld R3, #Chk_PassWord #256
3035/    1C5B : D6 04 AB                           call Bank_Call
3036/    1C5E : EB 03                              jr NZ, D_I_SprTbl
3037/    1C60 : D6 05 1F                           call Abort              ; password mismatch, go home
3038/    1C63 :                    ;
3039/    1C63 : D6 1A E3           D_I_SprTbl      call Ld_Param1
3040/    1C66 : 48 E0                              ld R4, R0               ; pass offset value
3041/    1C68 : 58 E1                              ld R5, R1               ; pass interleave value
3042/    1C6A : 2C 22                              ld R2, #Init_SprTbl /256
3043/    1C6C : 3C 9F                              ld R3, #Init_SprTbl #256
3044/    1C6E : D6 04 AB                           call Bank_Call
3045/    1C71 : 8D 12 CA                           jp Rd_Leave
3046/    1C74 :
3047/    1C74 :
3048/    1C74 :
```

```
3049/    1C74 :                     ;*********************************************************************
3050/    1C74 :                     ; Module Cmnd2.Assem  {continuation of the command processor module}
3051/    1C74 :                     ;
3052/    1C74 :                     ; This module contains all the code associated with processing
3053/    1C74 :                     ; the System commands.
3054/    1C74 :                     ;
3055/    1C74 :                     ;       PROCEDURE Sys_Read
3056/    1C74 :                     ;       PROCEDURE Sys_Write
3057/    1C74 :                     ;
3058/    1C74 :                     ;*********************************************************************
3059/    1C74 :
3060/    1C74 :                     ;*********************************************************************
3061/    1C74 :                     ;
3062/    1C74 :                     ; Procedure: Sys_Read
3063/    1C74 :                     ;
3064/    1C74 :                     ;  This procedure is the System Read command for Widget. It allows
3065/    1C74 :                     ;  the host to read up to 8 sequential blocks without sending any
3066/    1C74 :                     ;  additional commands. Between each block status is passed back to
3067/    1C74 :                     ;  the host with the data, and the host must set CMD before the next
3068/    1C74 :                     ;  block can be read. In general, the protocol is about the same as
3069/    1C74 :                     ;  in Pro_Read.
3070/    1C74 :                     ;
3071/    1C74 :                     ; Inputs: none
3072/    1C74 :                     ;
3073/    1C74 :                     ; Outputs: none
3074/    1C74 :                     ;
3075/    1C74 :                     ; Algorithm:
3076/    1C74 :                     ;
3077/    1C74 :                     ;  Begin
3078/    1C74 :                     ;    Ld_LgclBlk(Wid_Log_Offset)
3079/    1C74 :                     ;    Count:=Ld_Param1.FirstParam
3080/    1C74 :                     ;    Offset:=0
3081/    1C74 :                     ;    BlockNumber:=Load_Logical
3082/    1C74 :                     ;    ClrNormStat
3083/    1C74 :                     ;    Get_Type_Check(BlockNumber+Count)
3084/    1C74 :                     ;    Seek_Type:=Access
3085/    1C74 :                     ;    DiskStat.Wr_Op:=false
3086/    1C74 :                     ;    OverLap
3087/    1C74 :                     ;    If not(ReadBlock)
3088/    1C74 :                     ;      Then
3089/    1C74 :                     ;        SS_RdErr
3090/    1C74 :                     ;        Goto Pro_Rd_Exit
3091/    1C74 :                     ;    While Always Do
3092/    1C74 :                     ;      BlockNumber:=BlockNumber+1
3093/    1C74 :                     ;      If not(SrchCache)
3094/    1C74 :                     ;        Then OverLap
3095/    1C74 :                     ;      If not(Read_Fast)
3096/    1C74 :                     ;        Then Goto Pro_Rd_Exit
3097/    1C74 :                     ;      If Count>1
3098/    1C74 :                     ;        Then
3099/    1C74 :                     ;          Command_Pending, IBsy:=true, Response:=Sys_Rd_Resp
3100/    1C74 :                     ;          Rd_Leave2
3101/    1C74 :                     ;        Else RdLeave
3102/    1C74 :                     ;      Count:=Count-1
3103/    1C74 :                     ;  End
3104/    1C74 :                     ;
3105/    1C74 :                     ;*********************************************************************
3106/    1C74 :
3107/    1C74 : E6 0A 22            Sys_Read        ld Wrk_Io+10, #Sys_Rd_Resp
3108/    1C77 : D6 1C E3                            call Sys_SetUp
3109/    1C7A : E6 57 80            Sys_Rd_Seek     ld Seek_Type, #Access
3110/    1C7D : 56 56 DF                            and DiskStat, #0FFh-Wr_Op
3111/    1C80 : D6 1D 1D                            call Sys_Rw_Seek
3112/    1C83 : D6 1E A9                            call ReadBlock
3113/    1C86 : EB 11                               jr NZ, Sys_Rd_Next
3114/    1C88 : 8B 24                               jr Sys_RdErr
3115/    1C8A :                     ;
3116/    1C8A : D6 1D 2C            Sys_Rd_Lp       call Sys_Inc_Blk
3117/    1C8D : EB EB                               jr NZ, Sys_Rd_Seek
3118/    1C8F : 56 E0 3F                            and R0, #3Fh            ; mask off head value
3119/    1C92 : 09 55                               ld Sector, R0
3120/    1C94 : D6 1E C9            Sys_Rd_OvrLp    call Read_Fast
3121/    1C97 : 6B 15                               jr Z, Sys_RdErr
3122/    1C99 :                     ;
3123/    1C99 : 76 5A 02            Sys_Rd_Next     tm BlkStat, #B_Block    ; check if bad block came back!
3124/    1C9C : EB 29                               jr NZ, Sys_Rd_BB
3125/    1C9E : E6 0A 22            Sys_Rd_N1       ld Wrk_Io+10, #Sys_Rd_Resp
3126/    1CA1 : E6 0B C0                            ld Wrk_Io+11, #Cmnd_Pending+IBsy
3127/    1CA4 : 4A 03                               djnz R4, Sys_Rd_More
3128/    1CA6 : 8D 12 CA                            jp Rd_Leave
3129/    1CA9 :                     ;
3130/    1CA9 : D6 12 D2            Sys_Rd_More     call Rd_Leave1
3131/    1CAC : 8B DC                               jr Sys_Rd_Lp
3132/    1CAE :                     ;
3133/    1CAE : 70 E4               Sys_RdErr       push R4                 ; save block count
3134/    1CB0 : D6 12 F2                            call Read_Common        ; retry for exception handling
3135/    1CB3 : 50 E4               Sys_RdErr1      pop R4                  ; get block count
3136/    1CB5 : 70 FC                               push FLAGS              ; save result
3137/    1CB7 : 70 E0                               push R0                 ; save return code
3138/    1CB9 : D6 06 0D                            call Load_Logical       ; restore block number sequence
3139/    1CBC : 50 E0                               pop R0
3140/    1CBE : 50 FC                               pop FLAGS
3141/    1CC0 : EB D7                               jr NZ, Sys_Rd_Next
3142/    1CC2 : D6 16 B1                            call Data_Ex_Handler
3143/    1CC5 : 8B 07                               jr Sys_Chk_Ftl
3144/    1CC7 :                     ;
3145/    1CC7 : 70 E4               Sys_Rd_BB       push R4                 ; save block count
3146/    1CC9 : D6 12 DB                            call Pro_Rd_BB
3147/    1CCC : 8B E5                               jr Sys_RdErr1
3148/    1CCE :                     ;
3149/    1CCE : D6 1C D9            Sys_Chk_Ftl     call Chk_FatalStat
3150/    1CD1 : ED 12 CA                            jp NZ, Rd_Leave         ; get out if fatal error
3151/    1CD4 : D6 1E 5F                            call ClrNormStat
3152/    1CD7 : 8B C5                               jr Sys_Rd_N1
3153/    1CD9 :                     ;
3154/    1CD9 : 2C 14               Chk_FatalStat   ld R2, #CStatus0 /256
3155/    1CDB : 3C 95                               ld R3, #CStatus0 #256
3156/    1CDD : 82 02                               lde R0, @RR2            ; get 1st byte of standard stat
3157/    1CDF : 76 E0 01                            tm R0, #Op_Failed       ; check if operation failed
3158/    1CE2 : AF                                  ret
3159/    1CE3 :                     ;
3160/    1CE3 : D6 03 02            Sys_SetUp       call Ack_Read
3161/    1CE6 : D6 1E 5F                            call ClrNormStat
3162/    1CE9 : 0C 03                               ld R0, #Sys_Log_Offset
3163/    1CEB : D6 06 4E                            call Ld_LgclBlk
3164/    1CEE : D6 1A E3                            call Ld_Param1
3165/    1CF1 : 42 00                               or R0, R0               ; check for zero count
3166/    1CF3 : EB 03                               jr NZ, Sys_Set1
```

```
3167/    1CF5 : D6 05 1F                        call Abort
3168/    1CF8 : 48 E0            Sys_Set1       ld R4, R0               ; Count:=...
3169/    1CFA : D6 06 0D                        call Load_Logical
3170/    1CFD : 08 E4                           ld R0, R4
3171/    1CFF : 00 E0                           dec R0                  ; account for numbering 1..n
3172/    1D01 : 02 E0                           add R14, R0
3173/    1D03 : 16 ED 00                        adc R13, #0
3174/    1D06 : 16 EC 00                        adc R12, #0
3175/    1D09 : 2C 21                           ld R2, #Get_Type_Check /256
3176/    1D0B : 3C D2                           ld R3, #Get_Type_Check #256
3177/    1D0D : D6 04 AB                        call Bank_Call
3178/    1D10 : D6 06 0D                        call Load_Logical
3179/    1D13 : E6 58 02                        ld Data_Type, #User_Type
3180/    1D16 : E6 0C 10                        ld Wrk_Io+12, #WBuffer1 /256    ; get ready for more writes
3181/    1D19 : E6 0D 21                        ld Wrk_Io+13, #WBuffer1 #256
3182/    1D1C : AF                              ret
3183/    1D1D :                  ;
3184/    1D1D : 70 E4            Sys_Rw_Seek    push R4                 ; save state
3185/    1D1F : 2C 1F                           ld R2, #OverLap /256
3186/    1D21 : 3C 8D                           ld R3, #OverLap #256
3187/    1D23 : D6 04 AB                        call Bank_Call
3188/    1D26 : 50 E4                           pop R4
3189/    1D28 : D6 06 0D                        call Load_Logical
3190/    1D2B : AF                              ret
3191/    1D2C :                  ;
3192/    1D2C : 06 EE 01         Sys_Inc_Blk    add R14, #1             ; BlockNumber:=BlockNumber+1
3193/    1D2F : 16 ED 00                        adc R13, #0
3194/    1D32 : 16 EC 00                        adc R12, #0
3195/    1D35 : 2C 14                           ld R2, #LogicalBlock /256
3196/    1D37 : 3C A1                           ld R3, #LogicalBlock #256
3197/    1D39 : 1C 03                           ld R1, #3
3198/    1D3B : 0C 1C                           ld R0, #Wrk_Sys+12
3199/    1D3D : 93 02            Sys_Inc_Lp     ldei @RR2, @R0
3200/    1D3F : 1A FC                           djnz R1, Sys_Inc_Lp
3201/    1D41 : D6 16 5E                        call SrchCache          ; get the new physical adr
3202/    1D44 : 76 5A C0                        tm BlkStat, #CachSeek+CachHdChg ; check for head change
3203/    1D47 : AF                              ret
3204/    1D48 :
3205/    1D48 :
3206/    1D48 :                  ;*****************************************************************
3207/    1D48 :                  ;
3208/    1D48 :                  ; Procedure: Sys_Write
3209/    1D48 :                  ;
3210/    1D48 :                  ;  This procedure is the System Write command for Widget. In general
3211/    1D48 :                  ;  it is a major departure from the ProFile write protocol: Status is
3212/    1D48 :                  ;  not sent back to the host unless there is useful information in
3213/    1D48 :                  ;  the status -- the fact that there were no errors in a block
3214/    1D48 :                  ;  transfer is communicated to the host via Response Byte that is
3215/    1D48 :                  ;  handshaken across at CMD/BSY time. If an error has occurred, then
3216/    1D48 :                  ;  the host is to recognize this state and come back to read the
3217/    1D48 :                  ;  status from the controller, otherwise it is assumed that the next
3218/    1D48 :                  ;  thing that the controller wants is write data. Note that the
3219/    1D48 :                  ;  second handshake (data received acknowledgement) has been removed
3220/    1D48 :                  ;  from the protocol.
3221/    1D48 :                  ;
3222/    1D48 :                  ; Inputs: none
3223/    1D48 :                  ;
3224/    1D48 :                  ; Outputs: none
3225/    1D48 :                  ;
3226/    1D48 :                  ; Algorithm:
3227/    1D48 :                  ;
3228/    1D48 :                  ;  Begin
3229/    1D48 :                  ;    Ld_LgclBlk(Wid_Log_Offset)
3230/    1D48 :                  ;    Count:=Ld_Param1.FirstParam
3231/    1D48 :                  ;    BlockNumber:=Load_Logical
3232/    1D48 :                  ;    ClrNormStat
3233/    1D48 :                  ;    Get_Type_Check(BlockNumber+Count)
3234/    1D48 :                  ;    Seek_Type:=Access_Offset
3235/    1D48 :                  ;    DiskStat.Wr_Op:=true
3236/    1D48 :                  ;    OverLap
3237/    1D48 :                  ;    If not(ReadBlock)
3238/    1D48 :                  ;      Then
3239/    1D48 :                  ;        SS_RdErr
3240/    1D48 :                  ;        Goto Pro_Rd_Exit
3241/    1D48 :                  ;    While Always Do
3242/    1D48 :                  ;      BlockNumber:=BlockNumber+1
3243/    1D48 :                  ;      If not(SrchCache)
3244/    1D48 :                  ;        Then OverLap
3245/    1D48 :                  ;      If not(Write_Common)
3246/    1D48 :                  ;        Then Data_Ex_Handler(Wr_Common.Error.Code)
3247/    1D48 :                  ;      If not(Read_Fast)
3248/    1D48 :                  ;        Then Goto Pro_Rd_Exit
3249/    1D48 :                  ;        Else
3250/    1D48 :                  ;          If BlkStat.Bad_Block
3251/    1D48 :                  ;            Then Goto Pro_Rd_BB
3252/    1D48 :                  ;      If Count>1
3253/    1D48 :                  ;        Then
3254/    1D48 :                  ;          Command_Pending, IBsy:=true, Response:=Sys_Wr_Resp
3255/    1D48 :                  ;          Rd_Leave2
3256/    1D48 :                  ;        Else RdLeave
3257/    1D48 :                  ;      Count:=Count-1
3258/    1D48 :                  ;  End
3259/    1D48 :                  ;
3260/    1D48 :                  ;*****************************************************************
3261/    1D48 :
3262/    1D48 : E6 0A 23         Sys_Write      ld Wrk_Io+10, #Sys_Wr_Resp
3263/    1D4B : D6 1C E3                        call Sys_SetUp
3264/    1D4E : E6 57 90         Sys_Wr_Seek    ld Seek_Type, #Access+Offset
3265/    1D51 : 46 56 20                        or DiskStat, #Wr_Op
3266/    1D54 : D6 1D 1D                        call Sys_Rw_Seek
3267/    1D57 : 76 5A 02                        tm BlkStat, #B_Block    ; check if bad block
3268/    1D5A : EB 57                           jr NZ, Sys_Wr_BB
3269/    1D5C : D6 1D B8                        call WriteBlock
3270/    1D5F : EB 1C                           jr NZ, Sys_Wr_Next
3271/    1D61 : 8B 39                           jr Sys_WrErr
3272/    1D63 :                  ;
3273/    1D63 : D6 1D 2C         Sys_Wr_Lp      call Sys_Inc_Blk
3274/    1D66 : EB E6                           jr NZ, Sys_Wr_Seek
3275/    1D68 : 76 56 01                        tm DiskStat, #Offset_On ; check for offset on all writes
3276/    1D6B : 6B E1                           jr Z, Sys_Wr_Seek
3277/    1D6D : 56 E0 3F                        and R0, #3Fh            ; mask off head value
3278/    1D70 : 09 55                           ld Sector, R0
3279/    1D72 : 76 5A 02         Sys_Wr_OvrLp   tm BlkStat, #B_Block    ; check if bad block
3280/    1D75 : ED 1D B3                        jp NZ, Sys_Wr_BB
3281/    1D78 : D6 1D DC                        call Write_Fast
3282/    1D7B : 6B 1F                           jr Z, Sys_WrErr
3283/    1D7D :                  ;
3284/    1D7D : E6 0A 23         Sys_Wr_Next    ld Wrk_Io+10, #Sys_Wr_Resp
```

```
3285/    1D80 : E6 0B E0                        ld Wrk_Io+11, #Cmnd_Pending+IBsy+MultiWr
3286/    1D83 : E6 0C 10                        ld Wrk_Io+12, #WBuffer1 /256
3287/    1D86 : E6 0D 21                        ld Wrk_Io+13, #WBuffer1 #256
3288/    1D89 : 4A 0C                           djnz R4, Sys_Wr_More
3289/    1D8B :                        ;
3290/    1D8B : E6 0A 27                        ld Wrk_Io+10, #Sys_WrEnd_Resp
3291/    1D8E : E6 0B C0               Sys_Wr_HS       ld Wrk_Io+11, #Cmnd_Pending+IBsy
3292/    1D91 : D6 12 CF                        call Rd_Leave2
3293/    1D94 : 8D 12 CA                        jp Rd_Leave
3294/    1D97 :                        ;
3295/    1D97 : D6 02 5F               Sys_Wr_More     call Clr_Bsy
3296/    1D9A : 8B C7                           jr Sys_Wr_Lp
3297/    1D9C :                        ;
3298/    1D9C : D6 18 CF               Sys_WrErr       call Wr_Common          ; retry for exception handling
3299/    1D9F : EB DC                           jr NZ, Sys_Wr_Next
3300/    1DA1 :                        ;
3301/    1DA1 : D6 16 B1                        call Data_Ex_Handler
3302/    1DA4 : D6 1C D9               Sys_WrChk       call Chk_FatalStat
3303/    1DA7 : EB 05                           jr NZ, Sys_WrE1
3304/    1DA9 : D6 1E 5F                        call ClrNormStat
3305/    1DAC : 8B CF                           jr Sys_Wr_Next
3306/    1DAE : E6 0A A3               Sys_WrE1        ld Wrk_Io+10, #Sys_WrEx_Resp
3307/    1DB1 : 8B DB                           jr Sys_Wr_HS
3308/    1DB3 :                        ;
3309/    1DB3 : D6 18 C2               Sys_Wr_BB       call Wr_BBlock
3310/    1DB6 : 8B EC                           jr Sys_WrChk
3311/    1DB8 :
3312/    1DB8 :
3313/    1DB8 :
3314/    1DB8 :                        ;*****************************************************************
3315/    1DB8 :                        ; Module Write.Assem
3316/    1DB8 :                        ;
3317/    1DB8 :                        ;  This module contains all but the very most primitive of procedures
3318/    1DB8 :                        ;  (the routines that are resident are listed in the external spec's)
3319/    1DB8 :                        ;  that concern themselves with performing a write operation.
3320/    1DB8 :                        ;
3321/    1DB8 :                        ;       FUNCTION WriteBlock(Parent : BYTE {R8}) :
3322/    1DB8 :                        ;               BOOLEAN
3323/    1DB8 :                        ;               Status : BYTE {R0}
3324/    1DB8 :                        ;               WrErrCnt : BYTE {R1}
3325/    1DB8 :                        ;
3326/    1DB8 :                        ;*****************************************************************
3327/    1DB8 :
3328/    1DB8 :                        ;*****************************************************************
3329/    1DB8 :                        ;
3330/    1DB8 :                        ; Function: WriteBlock
3331/    1DB8 :                        ;
3332/    1DB8 :                        ;  This function assumes that the heads are positioned over the
3333/    1DB8 :                        ;  correct track and writes the block of data whose header matches
3334/    1DB8 :                        ;  the header that is made up of the cylinder, head and sector
3335/    1DB8 :                        ;  information that is stored in memory.
3336/    1DB8 :                        ;
3337/    1DB8 :                        ; Inputs: Parent : BYTE {R8}
3338/    1DB8 :                        ;
3339/    1DB8 :                        ; Outputs: WriteBlock : BOOLEAN {Zero flag, true if error in ReadBlock}
3340/    1DB8 :                        ;          Status     : BYTE {R0}
3341/    1DB8 :                        ;          WrErrCnt   : BYTE {R1}
3342/    1DB8 :                        ;
3343/    1DB8 :                        ; Global Variables Used: Cylinder, Head, Sector, Recovery
3344/    1DB8 :                        ;
3345/    1DB8 :                        ; Local Variables Used: WrRetryCnt : BYTE {R8}
3346/    1DB8 :                        ;                       WrError    : BOOLEAN {R9/bit 7}
3347/    1DB8 :                        ;                       WrExcept   : BOOLEAN {R9/bit 6}
3348/    1DB8 :                        ;                       WrSuccess  : BOOLEAN {R9/bit 5}
3349/    1DB8 :                        ;                       NoHeaderFound : BOOLEAN {R9/bit 4}
3350/    1DB8 :                        ;                       SectorsRead: BYTE {R10}
3351/    1DB8 :                        ;
3352/    1DB8 :                        ; Algorithm:
3353/    1DB8 :                        ;
3354/    1DB8 :                        ;  BEGIN
3355/    1DB8 :                        ;    SetDeadManTimer( WriteBlock, Parent )
3356/    1DB8 :                        ;    WrRetryCnt := 10
3357/    1DB8 :                        ;    WrErrCnt := 0
3358/    1DB8 :                        ;    WrError := False
3359/    1DB8 :                        ;    WrExcept := False
3360/    1DB8 :                        ;    NoHeaderFound := False
3361/    1DB8 :                        ;    REPEAT
3362/    1DB8 :                        ;      WHeader[ 1 ] := HiCylinder
3363/    1DB8 :                        ;      WHeader[ 2 ] := LoCylinder
3364/    1DB8 :                        ;      WHeader[ 3 ]/bits 7:6 := Head
3365/    1DB8 :                        ;      WHeader[ 3 ]/bits 5:0 := Sector
3366/    1DB8 :                        ;      WHeader[ 4 ] := Invert( WHeader[ 1 ] )
3367/    1DB8 :                        ;      WHeader[ 5 ] := Invert( WHeader[ 2 ] )
3368/    1DB8 :                        ;      WHeader[ 6 ] := Invert( WHeader[ 3 ] )
3369/    1DB8 :                        ;      WDataSync := $0001
3370/    1DB8 :                        ;     _
3371/    1DB8 :                        ;    /
3372/    1DB8 :                        ; R  |  Set-up external ram address counter for WRITE
3373/    1DB8 :                        ; E  |  Msel0:1 := Disk <--> Mem
3374/    1DB8 :                        ; S  |  WHILE SectorMark DO BEGIN END
3375/    1DB8 :                        ; I  |  StartL := True
3376/    1DB8 :                        ; D  |  WHILE NOT( SectorDnL ) DO BEGIN END
3377/    1DB8 :                        ; E  |  Status := Status_Port
3378/    1DB8 :                        ; N  |  StartL := False
3379/    1DB8 :                        ; T  |  Msel0:1 := Z8 <--> Mem
3380/    1DB8 :                        ;    \_
3381/    1DB8 :                        ;
3382/    1DB8 :                        ;      CASE Status.State OF
3383/    1DB8 :                        ;        NormalEndState        : WrStat.WrSuccessful := True
3384/    1DB8 :                        ;
3385/    1DB8 :                        ;        NoMatchingHeaderFound : WrStat.WrSuccessful := False
3386/    1DB8 :                        ;                                WrStat.WrError := True
3387/    1DB8 :                        ;                                WrStat.NoHeaderFound := True
3388/    1DB8 :                        ;
3389/    1DB8 :                        ;        AbnormalState         : Reset_StateMachine
3390/    1DB8 :                        ;                                Abort
3391/    1DB8 :                        ;
3392/    1DB8 :                        ;      IF Status.ServoErr OR NOT( Status.ServoRdy )
3393/    1DB8 :                        ;        THEN
3394/    1DB8 :                        ;          WrStat.WrError := True
3395/    1DB8 :                        ;          WrStat.Except := True
3396/    1DB8 :                        ;
3397/    1DB8 :                        ;      IF Status.WrtNvldL  AND WrSuccessful
3398/    1DB8 :                        ;        THEN
3399/    1DB8 :                        ;          WrStat.WrError := True
3400/    1DB8 :                        ;          WrStat.WrErrCnt := WrErrCnt + 1
3401/    1DB8 :                        ;          WrRetryCnt := WrRetryCnt - 1
3402/    1DB8 :                        ;        ELSE
```

```
3403/    1DB8 :                     ;          RdRetryCnt := 0
3404/    1DB8 :                     ;
3405/    1DB8 :                     ;    UNTIL NOT( Recovery ) OR NOT( WrError ) OR ( WrRetryCnt = 0 ) OR
3406/    1DB8 :                     ;                WrExcept OR ( NoHeaderFound )
3407/    1DB8 :                     ;    ClearDeadManTimer
3408/    1DB8 :                     ;  END
3409/    1DB8 :                     ;
3410/    1DB8 :                     ;*********************************************************************
3411/    1DB8 :
3412/    1DB8 : B0 E9               WriteBlock      clr R9                  ; clear booleans
3413/    1DBA : 8C 0A                               ld R8, #10              ; WrdRetryCnt:=10
3414/    1DBC : B0 EB                               clr R11                 ; ErrCnt:=0
3415/    1DBE : 46 56 20                            or DiskStat, #Wr_Op     ; make certain we are writing
3416/    1DC1 :                     ;
3417/    1DC1 : 2C 10               WrBlk_Rpt       ld R2, #WHeader /256    ; initialize gaps
3418/    1DC3 : 3C 0B                               ld R3, #WHeader #256
3419/    1DC5 : D6 05 E1                            call Load_Header
3420/    1DC8 :                     ;
3421/    1DC8 : 0C 00                               ld R0, #0
3422/    1DCA : 1C 0E                               ld R1, #(WDataSync - WDataGap)
3423/    1DCC : 92 02               WrGap_Lp        lde @RR2, R0            ; write 14x $00
3424/    1DCE : 3E                                  inc R3
3425/    1DCF : 1A FB                               djnz R1, WrGap_Lp
3426/    1DD1 :                     ;
3427/    1DD1 : 0C 01                               ld R0, #1
3428/    1DD3 : 92 02                               lde @RR2, R0            ; then first byte of sync
3429/    1DD5 : 3E                                  inc R3
3430/    1DD6 : 0C 00                               ld R0, #0               ; then second byte of sync
3431/    1DD8 : 92 02                               lde @RR2, R0
3432/    1DDA : 8B 1E                               jr Do_Write
3433/    1DDC :                     ;
3434/    1DDC : B0 E9               Write_Fast      clr R9                  ; clear booleans
3435/    1DDE : 8C 0A                               ld R8, #10              ; WrRetryCnt:=10
3436/    1DE0 : B0 EB                               clr R11                 ; ErrCnt:=0
3437/    1DE2 : 46 56 20                            or DiskStat, #Wr_Op     ; make certain we are writing
3438/    1DE5 : 2C 10                               ld R2, #(RHeader+2) /256        ; get Sector location
3439/    1DE7 : 3C 0D                               ld R3, #(RHeader+2) #256
3440/    1DE9 : 82 02                               lde R0, @RR2            ; get current Head/Sector
3441/    1DEB : 56 E0 C0                            and R0, #0C0h           ; mask old sector value
3442/    1DEE : 44 55 E0                            or R0, Sector           ; merge new sector value
3443/    1DF1 : 92 02                               lde @RR2, R0            ; store it in header
3444/    1DF3 : 06 E3 03                            add R3, #3              ; get location of Inverse Sector
3445/    1DF6 : 60 E0                               com R0
3446/    1DF8 : 92 02                               lde @RR2, R0            ; and store it!
3447/    1DFA :                     ;
3448/    1DFA : D6 03 0C            Do_Write        call Wr_Resident        ; go internal to the Z8
3449/    1DFD : 18 E0                               ld R1, R0               ; CASE Status.State
3450/    1DFF : 56 E1 0F                            and R1, #YMask
3451/    1E02 : 42 AA                               or R10, R10
3452/    1E04 : 6B 43                               jr Z, WrBlk_NoHdr
3453/    1E06 : A6 E1 02                            cp R1, #Norm_State      ; statemachine healthy?
3454/    1E09 : EB 36                               jr NZ, WrBlk_Abnorm
3455/    1E0B :                     ;
3456/    1E0B : 46 E9 20            WrBlk_Norm      or R9, #WrSuccess
3457/    1E0E : 18 E0                               ld R1, R0               ; IF ServorErr OR NOT(ServoRdy)
3458/    1E10 : 76 E1 10                            tm R1, #ServoErr
3459/    1E13 : EB 3F                               jr NZ, WrBlk_ServoErr
3460/    1E15 : 76 E1 20                            tm R1, #ServoRdy
3461/    1E18 : 6B 3A                               jr Z, WrBlk_ServoErr
3462/    1E1A : 76 E0 40            Wr_ServoOk      tm R0, #WrtNvldL        ; IF Status.WrtNtVldL (ECC error)?
3463/    1E1D : 6B 3A                               jr Z, WrBlk_NVld
3464/    1E1F :                     ;
3465/    1E1F : 56 E9 7F                            and R9, #0FFh-WrError
3466/    1E22 : 76 24 80            WrBlk_Until     tm Excpt_Stat, #Recovery
3467/    1E25 : 6B 07                               jr Z, WrBlk_End
3468/    1E27 : 76 E9 80                            tm R9, #WrError
3469/    1E2A : 6B 02                               jr Z, WrBlk_End
3470/    1E2C : 8A 0A                               djnz R8, WrBlk_Rpt1     ; try again if fault occurred
3471/    1E2E :                     ;
3472/    1E2E : B9 29               WrBlk_End       ld WrErrCnt, R11
3473/    1E30 : 08 E9                               ld R0, R9               ; send status back to caller
3474/    1E32 : 99 28                               ld WrStat, R9
3475/    1E34 : 66 E0 80                            tcm R0, #WrError        ; set zero flag if error
3476/    1E37 : AF                                  ret
3477/    1E38 :                     ;
3478/    1E38 : 2C 22               WrBlk_Rpt1      ld R2, #ZeroHeader /256 ; clear up header
3479/    1E3A : 3C 8B                               ld R3, #ZeroHeader #256
3480/    1E3C : D6 04 AB                            call Bank_Call
3481/    1E3F : 8B 80                               jr WrBlk_Rpt            ; and try again
3482/    1E41 :                     ;
3483/    1E41 : D6 05 AE            WrBlk_Abnorm    call Reset_StMach
3484/    1E44 : A8 E0                               ld R10, R0
3485/    1E46 : D6 05 1F                            call Abort
3486/    1E49 :                     ;
3487/    1E49 : 56 E9 DF            WrBlk_NoHdr     and R9, #0FFh-WrSuccess
3488/    1E4C : 46 E9 90            Wr_HdrErr       or R9, #WrError+WrNoHdrFnd
3489/    1E4F : 46 EB 20                            or R11, #20h
3490/    1E52 : 8B DA                               jr WrBlk_End
3491/    1E54 :                     ;
3492/    1E54 : 46 E9 C0            WrBlk_ServoErr  or R9, #WrError+WrSrvoErr       ; THEN WrError AND WrSrvoErr
3493/    1E57 : 8B D5                               jr WrBlk_End
3494/    1E59 :                     ;
3495/    1E59 : 46 E9 80            WrBlk_NVld      or R9, #WrError         ;  ELSE
3496/    1E5C : BE                                  inc R11
3497/    1E5D : 8B C3                               jr WrBlk_Until
3498/    1E5F :
3499/    1E5F :
3500/    1E5F :                     ;*********************************************************************
3501/    1E5F :                     ;
3502/    1E5F :                     ; Module: Utilities
3503/    1E5F :                     ;
3504/    1E5F :                     ;  This module contains all the routines that do not fall logically
3505/    1E5F :                     ;  into any specific grouping, yet are essential to the program.
3506/    1E5F :                     ;
3507/    1E5F :                     ;       PROCEDURE ClrNormStat
3508/    1E5F :                     ;       PROCEDURE Move4(Source : PTR {RR2}
3509/    1E5F :                     ;                       Destination : PTR {RR14})
3510/    1E5F :                     ;
3511/    1E5F :                     ;*********************************************************************
3512/    1E5F :
3513/    1E5F :
3514/    1E5F :                     ;*********************************************************************
3515/    1E5F :                     ;
3516/    1E5F :                     ; Procedure: ClrNormStat
3517/    1E5F :                     ;
3518/    1E5F :                     ;  This procedure zeroes the header buffer and the standard
3519/    1E5F :                     ;  status CStatus0. Spare table warning, power-on, and selftest
3520/    1E5F :                     ;  faults are preserved in the status bytes.
```

```
3521/    1E5F :                     ;
3522/    1E5F :                     ; Inputs: {none}
3523/    1E5F :                     ;
3524/    1E5F :                     ; Outputs: {none}
3525/    1E5F :                     ;
3526/    1E5F :                     ; Algorithm:
3527/    1E5F :                     ;
3528/    1E5F :                     ; BEGIN
3529/    1E5F :                     ;   FOR i := 1 TO Length( StandardStatus ) DO
3530/    1E5F :                     ;     StandardStatus[ i ] := 0
3531/    1E5F :                     ;     IF ( Excpt_Stat.SprTbl_Warn )
3532/    1E5F :                     ;       THEN SS_SprWarn
3533/    1E5F :                     ;     IF ( Excpt_Stat.PwrRst )
3534/    1E5F :                     ;       THEN
3535/    1E5F :                     ;         SetStatus( PowerReesest )
3536/    1E5F :                     ;         Excpt_Stat.PwrRst := False
3537/    1E5F :                     ;  END
3538/    1E5F :                     ;
3539/    1E5F :                     ;*********************************************************************
3540/    1E5F :
3541/    1E5F : 2C 14               ClrNormStat     ld R2, #CStatus0 /256
3542/    1E61 : 3C 95                               ld R3, #CStatus0 #256
3543/    1E63 : B0 E0                               clr R0
3544/    1E65 : 1C 04                               ld R1, #4               ; clear all status bytes
3545/    1E67 : 92 02               Clr_N_Stat      lde @RR2, R0
3546/    1E69 : A0 E2                               incw RR2
3547/    1E6B : 1A FA                               djnz R1, Clr_N_Stat
3548/    1E6D :                     ;
3549/    1E6D : 2C 22                               ld R2, #ZeroHeader /256 ; clear header
3550/    1E6F : 3C 8B                               ld R3, #ZeroHeader #256
3551/    1E71 : D6 04 AB                            call Bank_Call
3552/    1E74 :                     ;
3553/    1E74 : 76 24 40                            tm Excpt_Stat, #SprTbl_Warn
3554/    1E77 : 6B 03                               jr Z, Clr_N_PwrRst
3555/    1E79 : D6 05 A8                            call SS_SprWarn         ; preserve spare table warning
3556/    1E7C :                     ;
3557/    1E7C : 76 24 10            Clr_N_PwrRst    tm Excpt_Stat, #PwrRst
3558/    1E7F : 6B 0A                               jr Z, Clr_N_SlfTst
3559/    1E81 : 0C 02                               ld R0, #2               ; status byte 2
3560/    1E83 : 1C 80                               ld R1, #Power_Reset
3561/    1E85 : D6 04 03                            call SetStatus          ; report cold start
3562/    1E88 : 56 24 EF                            and Excpt_Stat, #0FFh-PwrRst
3563/    1E8B :                     ;
3564/    1E8B : 44 25 25            Clr_N_SlfTst    or SlfTst_Result, SlfTst_Result
3565/    1E8E : 6B 07                               jr Z, ClrNmStat3
3566/    1E90 : 0C 01                               ld R0, #1               ; status byte 1
3567/    1E92 : 1C 08                               ld R1, #Stat_SlfTst
3568/    1E94 : D6 04 03                            call SetStatus          ; report selftest errors
3569/    1E97 :                     ;
3570/    1E97 : 8D 04 F8            ClrNmStat3      jp Bank_Ret
3571/    1E9A :
3572/    1E9A :
3573/    1E9A :                     ;*********************************************************************
3574/    1E9A :                     ;
3575/    1E9A :                     ; Procedure: Move4_B0   {in Bank 0}
3576/    1E9A :                     ;
3577/    1E9A :                     ;  This procedure copies four bytes located in external RAM.
3578/    1E9A :                     ;
3579/    1E9A :                     ; Inputs: Src: PTR {RR2}
3580/    1E9A :                     ;         Dst: PTR {RR14}
3581/    1E9A :                     ;
3582/    1E9A :                     ; Outputs: none
3583/    1E9A :                     ;
3584/    1E9A :                     ; Global Variables Changed: RAM @RR14
3585/    1E9A :                     ;
3586/    1E9A :                     ; Local Variables Used: R1
3587/    1E9A :                     ;
3588/    1E9A :                     ; Algorithm:
3589/    1E9A :                     ;
3590/    1E9A :                     ;  BEGIN
3591/    1E9A :                     ;    FOR i := 0 TO 3 DO
3592/    1E9A :                     ;      @Destination( i ) := @Source( i )
3593/    1E9A :                     ;  END
3594/    1E9A :                     ;
3595/    1E9A :                     ;*********************************************************************
3596/    1E9A :
3597/    1E9A : 1C 04               Move4_B0        ld R1, #4               ; counter
3598/    1E9C : 82 02               Move4_B0_Lp     lde R0, @RR2            ; get from RR2
3599/    1E9E : 92 0E                               lde @RR14, R0           ; put into RR14
3600/    1EA0 : A0 E2                               incw RR2
3601/    1EA2 : A0 EE                               incw RR14
3602/    1EA4 : 1A F6                               djnz R1, Move4_B0_Lp
3603/    1EA6 : 8D 04 F8                            jp Bank_Ret
3604/    1EA9 :
3605/    1EA9 :
3606/    1EA9 :
3607/    1EA9 :                     ;*********************************************************************
3608/    1EA9 :                     ; Module Read.Assem
3609/    1EA9 :                     ;
3610/    1EA9 :                     ; This module contains all but the very most primitive of
3611/    1EA9 :                     ; procedures (the routines that are resident are listed
3612/    1EA9 :                     ; in the external spec's) that concern themselves with performing
3613/    1EA9 :                     ; a read operation.
3614/    1EA9 :                     ;
3615/    1EA9 :                     ;       FUNCTION ReadBlock(Parent : BYTE {R8}) :
3616/    1EA9 :                     ;               BOOLEAN
3617/    1EA9 :                     ;               Status : BYTE {R0}
3618/    1EA9 :                     ;               RdErrCnt : BYTE {R1}
3619/    1EA9 :                     ;
3620/    1EA9 :                     ;*********************************************************************
3621/    1EA9 :
3622/    1EA9 :                     ;*********************************************************************
3623/    1EA9 :                     ;
3624/    1EA9 :                     ; Function: ReadBlock
3625/    1EA9 :                     ;
3626/    1EA9 :                     ;  This function assumes that the heads are positioned over the
3627/    1EA9 :                     ;  correct track and reads the block of data whose header matched
3628/    1EA9 :                     ;  the header that is made up of the cylinder, head and sector
3629/    1EA9 :                     ;  information that is stored in memory.
3630/    1EA9 :                     ;
3631/    1EA9 :                     ; Inputs: Parent: BYTE {R8}
3632/    1EA9 :                     ;
3633/    1EA9 :                     ; Outputs: ReadBlock: BOOLEAN {zero flag, true if error in ReadBlock}
3634/    1EA9 :                     ;          Status:    BYTE {R0}
3635/    1EA9 :                     ;          RdErrCnt:  BYTE {R1}
3636/    1EA9 :                     ;
3637/    1EA9 :                     ; Global Variables Used: Cylinder, Head, Sector, Recovery
3638/    1EA9 :                     ;
```

```
3639/    1EA9 :                    ; Local Variables Used: RdRetryCnt:  BYTE {R8}
3640/    1EA9 :                    ;                       RdError:     BOOLEAN {R9/bit 7}
3641/    1EA9 :                    ;                       RdExcept:    BOOLEAN {R9/bit 6}
3642/    1EA9 :                    ;                       RdSuccess:   BOOLEAN {R9/bit 5}
3643/    1EA9 :                    ;                       SectorsRead: BOOLEAN {R9/bit 4}
3644/    1EA9 :                    ;
3645/    1EA9 :                    ; Algorithm:
3646/    1EA9 :                    ;
3647/    1EA9 :                    ;  Begin
3648/    1EA9 :                    ;    SetDeadManTimer(ReadBlock, Parent)
3649/    1EA9 :                    ;    RdRetryCnt:=10
3650/    1EA9 :                    ;    RdErrCnt:=0
3651/    1EA9 :                    ;    RdError:=false
3652/    1EA9 :                    ;    RdExcept:=false
3653/    1EA9 :                    ;    NoHeaderFound:=false
3654/    1EA9 :                    ;    SectorsRead:=2xNbrSctrs {try to find header for two rotations}
3655/    1EA9 :                    ;    Repeat
3656/    1EA9 :                    ;      RHeader[1]:=HiCylinder
3657/    1EA9 :                    ;      RHeader[2]:=LoCylinder
3658/    1EA9 :                    ;      RHeader[3]/bits 7:6:=Head
3659/    1EA9 :                    ;      RHeader[3]/bits 5:0:=Sector
3660/    1EA9 :                    ;      RHeader[4]:=invert(RHeader[1])
3661/    1EA9 :                    ;      RHeader[5]:=invert(RHeader[2])
3662/    1EA9 :                    ;      RHeader[6]:=invert(RHeader[3])
3663/    1EA9 :                    ;      ReadArray[RDummy-1]:=0
3664/    1EA9 :                    ;    /-
3665/    1EA9 :                    ;  R | Set-up external RAM address counter for read
3666/    1EA9 :                    ;  E | Msel0:1:=Disk<-->Mem
3667/    1EA9 :                    ;  S | While SectorMark Do Begin End
3668/    1EA9 :                    ;  I | StartL:=true
3669/    1EA9 :                    ;  D | While not(SectorDnL) Do Begin End
3670/    1EA9 :                    ;  E | Status:=StatusPort
3671/    1EA9 :                    ;  N | StartL:=false
3672/    1EA9 :                    ;  T | Msel0:1:=Z8<-->Mem
3673/    1EA9 :                    ;    \-
3674/    1EA9 :                    ;     Case Status.State Of
3675/    1EA9 :                    ;       NormalEndState:        RdSuccess:=true
3676/    1EA9 :                    ;       NoMatchingHeaderFound: RdSuccess:=false
3677/    1EA9 :                    ;                              RdError:=true
3678/    1EA9 :                    ;                              NoHeaderFound:=true
3679/    1EA9 :                    ;       AbnormalState:         Reset_StateMachine
3680/    1EA9 :                    ;                              Abort
3681/    1EA9 :                    ;     If Status.ServoErr Or not(Status.ServoRdy)
3682/    1EA9 :                    ;       Then
3683/    1EA9 :                    ;         RdError:=true
3684/    1EA9 :                    ;         RdExcept:=true
3685/    1EA9 :                    ;     If Status.CrcErr And RdSuccess
3686/    1EA9 :                    ;       Then
3687/    1EA9 :                    ;         RdError:=true
3688/    1EA9 :                    ;         RdErrCnt:=RdErrCnt+1
3689/    1EA9 :                    ;         RdRetryCnt:=RdRetryCnt-1
3690/    1EA9 :                    ;       Else
3691/    1EA9 :                    ;         If RdError
3692/    1EA9 :                    ;           Then
3693/    1EA9 :                    ;             BlockMove(Buffer2, RBuffer1)
3694/    1EA9 :                    ;             RdRetryCnt:=RdRetryCnt-1
3695/    1EA9 :                    ;    Until not(Recovery) Or not(RdError) Or RdRetryCnt=0 Or
3696/    1EA9 :                    ;            RdExcept Or NoHeaderFound
3697/    1EA9 :                    ;    ClearDeadManTimer
3698/    1EA9 :                    ;    Status:=R9
3699/    1EA9 :                    ;  End
3700/    1EA9 :                    ;
3701/    1EA9 :                    ;*****************************************************************
3702/    1EA9 :
3703/    1EA9 : B0 E9              ReadBlock       clr R9                  ; clear booleans
3704/    1EAB : 8C 0A                              ld R8, #10              ; RdRetryCnt:=10
3705/    1EAD : B0 EB                              clr R11                 ; ErrCnt:=0
3706/    1EAF : 56 56 DF                           and DiskStat, #0FFh-Wr_Op       ; make sure we are reading
3707/    1EB2 : 2C 10              RdBlk_Rpt       ld R2, #RHeader /256    ; initialize gaps
3708/    1EB4 : 3C 0B                              ld R3, #RHeader #256
3709/    1EB6 : D6 05 E1                           call Load_Header
3710/    1EB9 : 0C 00                              ld R0, #0
3711/    1EBB : 92 02                              lde @RR2, R0
3712/    1EBD : 2C 10                              ld R2, #(RDummy-1) /256
3713/    1EBF : 3C 17                              ld R3, #(RDummy-1) #256
3714/    1EC1 : 92 02                              lde @RR2, R0
3715/    1EC3 : A0 E2                              incw RR2
3716/    1EC5 : 92 02                              lde @RR2, R0
3717/    1EC7 : 8B 1E                              jr Do_Read
3718/    1EC9 :                    ;
3719/    1EC9 : B0 E9              Read_Fast       clr R9                  ; clear booleans
3720/    1ECB : 8C 0A                              ld R8, #10              ; RdRetryCnt:=10
3721/    1ECD : B0 EB                              clr R11                 ; ErrCnt:=0
3722/    1ECF : 56 56 DF                           and DiskStat, #0FFh-Wr_Op       ; make certain we are reading
3723/    1ED2 : 2C 10                              ld R2, #(RHeader+2) /256        ; get location of Sector
3724/    1ED4 : 3C 0D                              ld R3, #(RHeader+2) #256
3725/    1ED6 : 82 02                              lde R0, @RR2            ; get current head/sector value
3726/    1ED8 : 56 E0 C0                           and R0, #0C0h           ; mask out old sector value
3727/    1EDB : 44 55 E0                           or R0, Sector           ; merge in new sector
3728/    1EDE : 92 02                              lde @RR2, R0            ; and store it
3729/    1EE0 : 06 E3 03                           add R3, #3              ; get location of inverse head/sector
3730/    1EE3 : 60 E0                              com R0
3731/    1EE5 : 92 02                              lde @RR2, R0            ; and store it, too!
3732/    1EE7 : D6 03 46           Do_Read         call Rd_Resident        ; go internal to the Z8
3733/    1EEA : 18 E0                              ld R1, R0               ; Case Status.State
3734/    1EEC : 56 E1 0F                           and R1, #YMask
3735/    1EEF : 42 AA                              or R10, R10
3736/    1EF1 : 6B 49                              jr Z, RdBlk_NoHdr
3737/    1EF3 : A6 E1 02                           cp R1, #Norm_State
3738/    1EF6 : EB 3C                              jr NZ, RdBlk_AbNorm
3739/    1EF8 :                    ;
3740/    1EF8 : 46 E9 20           RdBlk_Norm      or R9, #RdSuccess
3741/    1EFB : 18 E0                              ld R1, R0               ; If ServoErr Or not(ServoRdy)
3742/    1EFD : 76 E1 10                           tm R1, #ServoErr
3743/    1F00 : EB 57                              jr NZ, RD_ServoErr
3744/    1F02 : 76 E1 20                           tm R1, #ServoRdy
3745/    1F05 : 6B 52                              jr Z, RD_ServoErr
3746/    1F07 :                    ;
3747/    1F07 : 76 E0 80           Rd_ServoOk      tm R0, #CrcErrL         ; If Status.CrcErr
3748/    1F0A : 6B 52                              jr Z, RD_BadCrc
3749/    1F0C : 76 E0 40                           tm R0, #WrtNvldL        ; Or Status.EccErr
3750/    1F0F : 6B 6D                              jr Z, RD_BadEcc
3751/    1F11 :                    ;
3752/    1F11 : 76 E9 80           Rd_NoCrcErr     tm R9, #RdError         ; Then If RdError
3753/    1F14 : EB 6E                              jr NZ, RdBlk_Rmove
3754/    1F16 :                    ;
3755/    1F16 : 76 24 80           RdBlk_Until     tm Excpt_Stat, #Recovery
3756/    1F19 : 6B 05                              jr Z, RdBlk_End
```

```
3757/    1F1B : 76 E9 80                          tm R9, #RdError
3758/    1F1E : EB 0A                             jr NZ, RdB_Rpt1
3759/    1F20 :                         ;
3760/    1F20 : B9 27                   RdBlk_End ld RdErrCnt, R11
3761/    1F22 : 08 E9                             ld R0, R9               ; send status back to caller
3762/    1F24 : 99 26                             ld RdStat, R9
3763/    1F26 : 66 E0 80                          tcm R0, #RdError        ; set zero flag if error
3764/    1F29 : AF                                ret
3765/    1F2A :                         ;
3766/    1F2A : 2C 22                   RdB_Rpt1  ld R2, #ZeroHeader /256
3767/    1F2C : 3C 8B                             ld R3, #ZeroHeader #256
3768/    1F2E : D6 04 AB                          call Bank_Call
3769/    1F31 : 8D 1E B2                          jp RdBlk_Rpt
3770/    1F34 :                         ;
3771/    1F34 : D6 05 AE                RdBlk_AbNorm call Reset_StMach
3772/    1F37 : A8 E0                             ld R10, R0
3773/    1F39 : D6 05 1F                          call Abort
3774/    1F3C :                         ;
3775/    1F3C : 56 E9 DF                RdBlk_NoHdr and R9, #0FFh-RdSuccess
3776/    1F3F : 46 E9 80                          or R9, #Error
3777/    1F42 : 76 EB 20                          tm R11, #Hdr_MisMatch
3778/    1F45 : EB 0D                             jr NZ, Rd_HdrErr
3779/    1F47 : 46 EB 20                          or R11, #Hdr_MisMatch
3780/    1F4A : 76 24 80                          tm Excpt_Stat, #Recovery        ; auto offset ONLY if recovery on
3781/    1F4D : 6B C7                             jr Z, RdBlk_Until
3782/    1F4F : D6 05 CA                          call ReSeek             ; set auto-offset
3783/    1F52 : 8B C2                             jr RdBlk_Until
3784/    1F54 :                         ;
3785/    1F54 : 46 E9 90                Rd_HdrErr or R9, #RdNoHdrFnd+RdError
3786/    1F57 : 8B C7                             jr RdBlk_End
3787/    1F59 :                         ;
3788/    1F59 : 46 E9 C0                RD_ServoErr or R9, #RdError+RdSrvoErr       ; Then RdError And RdServoErr
3789/    1F5C : 8B C2                             jr RdBlk_End
3790/    1F5E :                         ;
3791/    1F5E : 46 E9 88                RD_BadCrc or R9, #RdError+RdCrcErr        ; Else
3792/    1F61 : BE                                inc R11
3793/    1F62 : 46 EB 40                          or R11, #CrcStat
3794/    1F65 : 76 E0 40                          tm R0, #WrtNvldL        ; check for ECC error, too
3795/    1F68 : EB 10                             jr NZ, Rd_B_Crc_1
3796/    1F6A : 46 EB 80                RD_Bad1   or R11, #EccStat
3797/    1F6D : 76 24 80                          tm Excpt_Stat, #Recovery        ; re-seek only if recovery is on
3798/    1F70 : 6B 08                             jr Z, Rd_B_Crc_1
3799/    1F72 : 76 56 01                          tm DiskStat, #Offset_On         ; set auto-offset if needed
3800/    1F75 : EB 03                             jr NZ, Rd_B_Crc_1
3801/    1F77 : D6 05 CA                          call ReSeek
3802/    1F7A : 8A 9A                   Rd_B_Crc_1 djnz R8, RdBlk_Until
3803/    1F7C : 8B A2                             jr RdBlk_End
3804/    1F7E :                         ;
3805/    1F7E : 46 E9 88                RD_BadEcc  or R9, #RdError+RdCrcErr
3806/    1F81 : BE                                inc R11
3807/    1F82 : 8B E6                             jr RD_Bad1
3808/    1F84 :                         ;
3809/    1F84 : 2C 20                   RdBlk_Rmove ld R2, #RBuf_To_Buf2 /256
3810/    1F86 : 3C 99                             ld R3, #RBuf_To_Buf2 #256
3811/    1F88 : D6 04 AB                          call Bank_Call
3812/    1F8B : 8B ED                             jr Rd_B_Crc_1
3813/    1F8D :
3814/    1F8D :
3815/    1F8D :
3816/    1F8D :                         ;*****************************************************************
3817/    1F8D :                         ; Module Srvo0.Assem
3818/    1F8D :                         ;
3819/    1F8D :                         ; This module contains all the specifications and source code for
3820/    1F8D :                         ; controlling the Widget Servo Board (i.e. communication
3821/    1F8D :                         ; protocol, seek and head positioning, spare table lookup, etc).
3822/    1F8D :                         ;
3823/    1F8D :                         ;*****************************************************************
3824/    1F8D :
3825/    1F8D :                         ;*****************************************************************
3826/    1F8D :                         ;
3827/    1F8D :                         ; Function: OverLap  {overlapped seek}
3828/    1F8D :                         ;
3829/    1F8D :                         ;  This fuction allows the drive to begin a seek operation (if
3830/    1F8D :                         ;  one is needed) before getting tied up with the SOS driver.
3831/    1F8D :                         ;
3832/    1F8D :                         ; Inputs: BlockNumber: 3 BYTES {R12:14}
3833/    1F8D :                         ;
3834/    1F8D :                         ; Outputs: Block Status: BYTE {R0}
3835/    1F8D :                         ;
3836/    1F8D :                         ; Local Variables: Retry: BYTE {R4}
3837/    1F8D :                         ;
3838/    1F8D :                         ; Algorithm:
3839/    1F8D :                         ;
3840/    1F8D :                         ;  BEGIN
3841/    1F8D :                         ;    IF ( Srch_Cache( BlockNumber ) )
3842/    1F8D :                         ;       THEN
3843/    1F8D :                         ;         IF ( Srch_Cache.Head <> Head )
3844/    1F8D :                         ;           THEN SelectHead( Srch_Cache.Head )
3845/    1F8D :                         ;       ELSE
3846/    1F8D :                         ;         PositionHeads( NoWait, CnvrtLogical( BlockNumber ) )
3847/    1F8D :                         ;         Load_Cache
3848/    1F8D :                         ;         BlkStat := CachStat[ 0 ]
3849/    1F8D :                         ;         Seek( CnvrtLogical( BlockNumber ) )
3850/    1F8D :                         ;   END
3851/    1F8D :                         ;
3852/    1F8D :                         ;*****************************************************************
3853/    1F8D :
3854/    1F8D : D6 16 5E                OverLap   call SrchCache
3855/    1F90 : EB 2A                             jr NZ, OvrLapSeek
3856/    1F92 : 70 E0                             push R0                 ; save Head/Sector info
3857/    1F94 : 76 5A 40                          tm BlkStat, #CachHdChg  ; check for a head change
3858/    1F97 : 6B 1A                             jr Z, OverLd_Sctr
3859/    1F99 :                         ;
3860/    1F99 : B6 54 01                          xor Head, #1            ; complement the head value
3861/    1F9C : E8 54                             ld R14, Head
3862/    1F9E : 2C 27                             ld R2, #SelectHead /256
3863/    1FA0 : 3C 17                             ld R3, #SelectHead #256
3864/    1FA2 : D6 04 AB                          call Bank_Call
3865/    1FA5 :                         ;
3866/    1FA5 : 2C 2B                   Ovr_LdCache ld R2, #Load_Cache /256
3867/    1FA7 : 3C 41                             ld R3, #Load_Cache #256
3868/    1FA9 : D6 04 AB                          call Bank_Call          ; update the cache
3869/    1FAC : 2C 27                             ld R2, #Chk_Offset /256
3870/    1FAE : 3C F7                             ld R3, #Chk_Offset #256
3871/    1FB0 : D6 04 AB                          call Bank_Call
3872/    1FB3 :                         ;
3873/    1FB3 : 50 E3                   OverLd_Sctr pop R3                 ; get Head/Sector infor from stack
3874/    1FB5 : 56 E3 1F                          and R3, #1Fh            ; mask off all but sector info
```

```
3875/    1FB8 : 39 55                              ld Sector, R3
3876/    1FBA : 8B 28                              jr OvrLp_End
3877/    1FBC :                        ;
3878/    1FBC : 2C 16                  OvrLapSeek  ld R2, #CnvrtLogical /256
3879/    1FBE : 3C 3A                              ld R3, #CnvrtLogical #256
3880/    1FC0 : D6 04 AB                           call Bank_Call
3881/    1FC3 : B0 E8                  OvrLp_Lp    clr R8                  ; no wait
3882/    1FC5 : 2C 26                              ld R2, #PosHeads /256
3883/    1FC7 : 3C 88                              ld R3, #PosHeads #256
3884/    1FC9 : D6 04 AB                           call Bank_Call
3885/    1FCC : D6 10 97               OvrLp_2     call ExtPush_Vector
3886/    1FCF : 2C 2B                              ld R2, #Load_Cache /256
3887/    1FD1 : 3C 41                              ld R3, #Load_Cache #256
3888/    1FD3 : D6 04 AB                           call Bank_Call          ; update the cache
3889/    1FD6 : 2C 16                              ld R2, #CacheStat /256
3890/    1FD8 : 3C E8                              ld R3, #CacheStat #256
3891/    1FDA : 82 02                              lde R0, @RR2
3892/    1FDC : 09 5A                              ld BlkStat, R0
3893/    1FDE : D6 10 9F               OvrLp_S_1   call ExtPop_Vector
3894/    1FE1 : D6 05 D7                           call New_Seek
3895/    1FE4 : 8D 04 F8               OvrLp_End   jp Bank_Ret
3896/    1FE7 :
3897/    1FE7 : FF FF FF FF FF FF                  DB 02000h-$ dup(0FFh)   ; pad with $FF's
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF
3898/    2000 :                        ;=================================================================
3899/    2000 :                        ;=================================================================
3900/    2000 :
3901/    2000 :                        ;*****************************************************************
3902/    2000 :                        ; Module Bank1.Assem
3903/    2000 :                        ;
3904/    2000 :                        ;*****************************************************************
3905/    2000 :                                org 1000h
3906/    2000 :                                phase 2000h             ; EPROM will start at 4k so 4k 01000h offset
3907/    2000 : EF 78                            DW Checksum1
3908/    2002 : 01                               DB 1                    ; bank 1
3909/    2003 : F0 78 3C 1E                      DW Passwrd1, Passwrd2
3910/    2007 :
3911/    2007 :
3912/    2007 :
3913/    2007 :                        ;*****************************************************************
3914/    2007 :                        ; Module RdHdr.B1.Assem
3915/    2007 :                        ;
3916/    2007 :                        ;  This module contains all but the very most primitive of of
3917/    2007 :                        ;  procedures (the routines that are resident are listed in the
3918/    2007 :                        ;  external spec's) that concern themselves with performing a read
3919/    2007 :                        ;  operation.
3920/    2007 :                        ;
3921/    2007 :                        ;       FUNCTION ReadHdr(Parent : BYTE {R8}) :
3922/    2007 :                        ;                BOOLEAN
3923/    2007 :                        ;                Status : BYTE {R0}
3924/    2007 :                        ;
3925/    2007 :                        ;*****************************************************************
3926/    2007 :
3927/    2007 :                        ;*****************************************************************
3928/    2007 :                        ;
3929/    2007 :                        ; Function: ReadHdr
3930/    2007 :                        ;
3931/    2007 :                        ;  This function assumes that the heads are positioned over the
3932/    2007 :                        ;  correct track and reads the block of data following the
3933/    2007 :                        ;  next sector mark. This routine is used for two reasons: 1) to
3934/    2007 :                        ;  verify the header that is laid out on a block, and 2) to try
3935/    2007 :                        ;  to recover the data within a block if the header is munched.
3936/    2007 :                        ;
3937/    2007 :                        ; Inputs: Parent: BYTE {R8}
3938/    2007 :                        ;
3939/    2007 :                        ; Outputs: ReadHdr: BOOLEAN {zero flag, true if error in ReadHdr}
3940/    2007 :                        ;          Status:  BYTE {R0}
3941/    2007 :                        ;
3942/    2007 :                        ; Global Variables Used: Cylinder, Head, Sector
3943/    2007 :                        ;
3944/    2007 :                        ; Local Variables Used: RdHError:  BOOLEAN {R9/bit 7}
3945/    2007 :                        ;                       RdHExcept: BOOLEAN {R9/bit 6}
3946/    2007 :                        ;
3947/    2007 :                        ; Algorithm:
3948/    2007 :                        ;
3949/    2007 :                        ;  Begin
3950/    2007 :                        ;    SetDeadManTimer(ReadBlock, Parent)
3951/    2007 :                        ;    RdHError:=false
3952/    2007 :                        ;    RdHExcept:=false
3953/    2007 :                        ;    RdHSctrGap:=0
3954/    2007 :                        ;    RdHHdrGap:=0
3955/    2007 :                        ;   /-
3956/    2007 :                        ; R | Set-up external RAM address counter for read header
3957/    2007 :                        ; E | Msel0:1:=Disk<-->Mem
3958/    2007 :                        ; S | While SectorMark Do Begin End
3959/    2007 :                        ; I | StartL:=true
3960/    2007 :                        ; D | While not(SectorDnL) Do Begin End
3961/    2007 :                        ; E | Status:=StatusPort
3962/    2007 :                        ; N | StartL:=false
3963/    2007 :                        ; T | Msel0:1:=Z8<-->Mem
3964/    2007 :                        ;   \-
3965/    2007 :                        ;    If Status.State<>NormalEndState
3966/    2007 :                        ;      Then
3967/    2007 :                        ;        Reset_StateMachine
3968/    2007 :                        ;        Abort
3969/    2007 :                        ;    If Status.ServoErr or not(Status.ServoRdy)
3970/    2007 :                        ;      Then
3971/    2007 :                        ;        RdHError:=true
3972/    2007 :                        ;        RdHExcept:=true
3973/    2007 :                        ;    If Status.CrcErrL then RdHError:=true
3974/    2007 :                        ;    ClearDeadManTimer
3975/    2007 :                        ;    Status/bit7:=RdHError
3976/    2007 :                        ;    Status/bit7:=RdHExcept
3977/    2007 :                        ;  End
3978/    2007 :                        ;
3979/    2007 :                        ;*****************************************************************
3980/    2007 :
3981/    2007 : 56 56 DF               ReadHdr     and DiskStat, #0FFh-Wr_Op       ; make sure we are reading
3982/    200A : B0 E9                              clr R9                  ; clear booleans
3983/    200C : D6 03 40                           call RdHdr_Resident     ; go internal to the Z8
3984/    200F : 18 E0                              ld R1, R0               ; If Status.State
3985/    2011 : 56 E1 0F                           and R1, #YMask
3986/    2014 : A6 E1 02                           cp R1, #Norm_State
3987/    2017 : 6B 0D                              jr Z, RdHdr_Norm
3988/    2019 :                        ;
```

```
3989/    2019 : D6 05 AE                        call Reset_StMach
3990/    201C : 56 56 BF                        and DiskStat, #0FFh-RdHdrRecal
3991/    201F : B8 EA                           ld R11, R10
3992/    2021 : A8 E0                           ld R10, R0
3993/    2023 : D6 05 1F                        call Abort
3994/    2026 :                    ;
3995/    2026 : 18 E0              RdHdr_Norm   ld R1, R0               ; If ServoErr or not(ServoRdy)
3996/    2028 : 76 E1 10                        tm R1, #ServoErr
3997/    202B : EB 05                           jr NZ, RdHd_ServoErr
3998/    202D : 76 E1 20                        tm R1, #ServoRdy
3999/    2030 : EB 05                           jr NZ, RdHd_SrvoOk
4000/    2032 :                    ;
4001/    2032 : 46 E9 C0           RdHd_ServoErr or R9, #RdHError+RdHSrvoErr     ; Then RdHError and RdHServoErr
4002/    2035 : 8B 1A                           jr RdH_End
4003/    2037 :                    ;
4004/    2037 : 76 E0 80           RdHd_SrvoOk  tm R0, #CrcErrL         ; If Status.CrcErr
4005/    203A : 6B 0D                           jr Z, RdH_CrcErr
4006/    203C :                    ;
4007/    203C : 76 E0 40           RdH_ChkEcc   tm R0, #WrtNvldL        ; or Status.EccErr
4008/    203F : EB 10                           jr NZ, RdH_End
4009/    2041 : 46 E9 88                        or R9, #RdHError+RdCrcErr
4010/    2044 : 46 27 80                        or RdErrCnt, #EccStat
4011/    2047 : 8B 08                           jr RdH_End
4012/    2049 :                    ;
4013/    2049 : 46 E9 88           RdH_CrcErr   or R9, #RdHError+RdCrcErr       ; Else
4014/    204C : E6 27 40                        ld RdErrCnt, #CrcStat
4015/    204F : 8B EB                           jr RdH_ChkEcc
4016/    2051 :                    ;
4017/    2051 : 08 E9              RdH_End      ld R0, R9               ; send status back to caller
4018/    2053 : 99 26                           ld RdStat, R9
4019/    2055 : 66 E0 80                        tcm R0, #RdHError       ; set zero flag if error
4020/    2058 : 8D 04 F8                        jp Bank_Ret
4021/    205B :
4022/    205B :
4023/    205B :
4024/    205B :                    ;*********************************************************************
4025/    205B :                    ; Module Utils.B1
4026/    205B :                    ;
4027/    205B :                    ; This module contains routines that are part of the main utility
4028/    205B :                    ; module and that must be kept in Bank 1.
4029/    205B :                    ;
4030/    205B :                    ;       PROCEDURE RBuf_To_Buf2
4031/    205B :                    ;       PROCEDURE Buf2_To_RBuf
4032/    205B :                    ;       PROCEDURE WrBuf_To_Buf2
4033/    205B :                    ;       PROCEDURE Buf2_To_WrBuf
4034/    205B :                    ;       PROCEDURE RBuf_To_Spr
4035/    205B :                    ;       PROCEDURE Spr_To_RBuf
4036/    205B :                    ;       PROCEDURE WrBuf_To_Spr
4037/    205B :                    ;       PROCEDURE Spr_To_WrBuf
4038/    205B :                    ;       PROCEDURE Zero_RdBuf
4039/    205B :                    ;       PROCEDURE Zero_Fmt
4040/    205B :                    ;       FUNCTION Get_Cyl_H_S(PhysicalBlock : 3 BYTES {RR12}) :
4041/    205B :                    ;               Cylinder : WORD {RR12}
4042/    205B :                    ;               Head : BYTE {R14}
4043/    205B :                    ;               Sector : BYTE {R15}
4044/    205B :                    ;       FUNCTION GoodHdr : BOOLEAN
4045/    205B :                    ;               Cylinder : WORD {RR0}
4046/    205B :                    ;               Head : BYTE {R2}
4047/    205B :                    ;               Sector : BYTE {R3}
4048/    205B :                    ;       PROCEDURE BlockMove(Destination : PTR {RR2}
4049/    205B :                    ;               Source : PTR {RR0})
4050/    205B :                    ;               FUNCTION UpDate_Hdr : BOOLEAN
4051/    205B :                    ;               FUNCTION Get_Type( DriverType : BYTE {R8}) :
4052/    205B :                    ;>                      BlockType : 3 BITS {R8/bits 3:1}
4053/    205B :                    ;>                      BlockNumber : 3 BYTES {R12:14}
4054/    205B :                    ;>
4055/    205B :                    ;*********************************************************************
4056/    205B :
4057/    205B :                    ;*********************************************************************
4058/    205B :                    ;
4059/    205B :                    ; Procedures:   Ld_Stand_Stat  {move standard status to output buffer}
4060/    205B :                    ;               RBuf_To_Buf2   {move ReadBuffer to Buffer2}
4061/    205B :                    ;               Buf2_To_RBuf   {move Buffer2 to ReadBuffer}
4062/    205B :                    ;               WrBuf_To_Buf2  {move WriteBuffer to Buffer2}
4063/    205B :                    ;               Buf2_To_WrBuf  {move Buffer2 to WriteBuffer}
4064/    205B :                    ;               RBuf_To_Spr    {move ReadBuffer to SpareArray}
4065/    205B :                    ;               pr_To_RBuf     {move SpareArray to ReadBuffer}
4066/    205B :                    ;               Zero_RdBuf     {move zeros to ReadBuffer}
4067/    205B :                    ;
4068/    205B :                    ; Inputs: none
4069/    205B :                    ;
4070/    205B :                    ; Outputs: none
4071/    205B :                    ;
4072/    205B :                    ;*********************************************************************
4073/    205B :
4074/    205B : 0C 12              Buf2_To_RBuf ld R0, #Buf2Array /256
4075/    205D : 1C 74                           ld R1, #Buf2Array #256
4076/    205F : 2C 10                           ld R2, #RDummy /256
4077/    2061 : 3C 18                           ld R3, #RDummy #256
4078/    2063 : 8B 41                           jr B_Move
4079/    2065 :                    ;
4080/    2065 : A6 58 02           WrBuf_To_Buf2 cp Data_Type, #User_Type        ; nop if sparetable data
4081/    2068 : EB 3F                           jr NZ, Return_Vector
4082/    206A : 0C 10                           ld R0, #(WBuffer1-1) /256
4083/    206C : 1C 20                           ld R1, #(WBuffer1-1) #256
4084/    206E : 8B 32                           jr X_To_Buf2
4085/    2070 :                    ;
4086/    2070 : A6 58 02           Buf2_To_WrBuf cp Data_Type, #User_Type
4087/    2073 : EB 37                           jr NZ, Spr_To_WrBuf
4088/    2075 : 0C 12                           ld R0, #Buf2Array /256
4089/    2077 : 1C 74                           ld R1, #Buf2Array #256
4090/    2079 : 2C 10                           ld R2, #(WBuffer1-1) /256
4091/    207B : 3C 20                           ld R3, #(WBuffer1-1) #256
4092/    207D : 8B 27                           jr B_Move
4093/    207F :                    ;
4094/    207F : 0C 14              Spr_To_RBuf  ld R0, #SpareArray /256
4095/    2081 : 1C BB                           ld R1, #SpareArray #256
4096/    2083 : 2C 10                           ld R2, #RBuffer1 /256
4097/    2085 : 3C 19                           ld R3, #RBuffer1 #256
4098/    2087 : 8B 1D                           jr B_Move
4099/    2089 :
4100/    2089 : 0C 10              RBuf_To_Spr  ld R0, #RBuffer1 /256
4101/    208B : 1C 19                           ld R1, #RBuffer1 #256
4102/    208D : 2C 14              X_To_Spr     ld R2, #SpareArray /256
4103/    208F : 3C BB                           ld R3, #SpareArray #256
4104/    2091 : 8B 13                           jr B_Move
4105/    2093 :
4106/    2093 : 0C 10              WrBuf_To_Spr ld R0, #WBuffer1 /256
```

```
4107/    2095 : 1C 21                              ld R1, #WBuffer1 #256
4108/    2097 : 8B F4                              jr X_To_Spr
4109/    2099 :
4110/    2099 : A6 58 02           RBuf_To_Buf2    cp Data_Type, #User_Type        ; nop if sparetable data
4111/    209C : EB 0B                              jr NZ, Return_Vector
4112/    209E : 0C 10                              ld R0, #RDummy /256
4113/    20A0 : 1C 18                              ld R1, #RDummy #256
4114/    20A2 : 2C 12              X_To_Buf2       ld R2, #Buf2Array /256
4115/    20A4 : 3C 74                              ld R3, #Buf2Array #256
4116/    20A6 : D6 21 4F           B_Move          call BlockMove
4117/    20A9 : 8D 04 F8           Return_Vector   jp Bank_Ret
4118/    20AC :
4119/    20AC :
4120/    20AC : 0C 14              Spr_To_WrBuf    ld R0, #SpareArray /256
4121/    20AE : 1C BB                              ld R1, #SpareArray #256
4122/    20B0 : 2C 10                              ld R2, #WBuffer1 /256
4123/    20B2 : 3C 21                              ld R3, #WBuffer1 #256
4124/    20B4 : 8B F0                              jr B_Move
4125/    20B6 : 8B F1              Zero_RdBuf      jr Return_Vector                ; nop (taken out to save space)
4126/    20B8 :
4127/    20B8 : 2C 10              Zero_Fmt        ld R2, #ReadArray /256
4128/    20BA : 3C 00                              ld R3, #ReadArray #256
4129/    20BC : D6 22 7A                           call ZeroBlock
4130/    20BF : 2C 10                              ld R2, #FBuffer1 /256
4131/    20C1 : 3C 52                              ld R3, #FBuffer1 #256
4132/    20C3 : EC 02                              ld R14, #(FEndGap-FBuffer1) /256
4133/    20C5 : FC 1C                              ld R15, #(FEndGap-FBuffer1) #256
4134/    20C7 : 0C C6                              ld R0, #0C6h                    ; init block to pattern
4135/    20C9 : 92 02              Zero_Flp1       lde @RR2, R0
4136/    20CB : A0 E2                              incw RR2
4137/    20CD : 80 EE                              decw RR14
4138/    20CF : EB F8                              jr NZ, Zero_Flp1
4139/    20D1 : 8D 20 A9                           jp Return_Vector
4140/    20D4 :
4141/    20D4 :
4142/    20D4 :                    ;*****************************************************************
4143/    20D4 :                    ;
4144/    20D4 :                    ; Function: Get_Cyl_H_S  {get cylinder, head, and sector}
4145/    20D4 :                    ;
4146/    20D4 :                    ;  This routine extracts the cylinder, head, and sector information
4147/    20D4 :                    ;  from a physical block number.
4148/    20D4 :                    ;
4149/    20D4 :                    ; Inputs: PhysicalBlockNumber: 3 BYTES {R12:14}
4150/    20D4 :                    ;
4151/    20D4 :                    ; Outputs: HiCylinder: BYTE {R12}
4152/    20D4 :                    ;          LoCylinder: BYTE {R13}
4153/    20D4 :                    ;          Head:       BYTE {R14}
4154/    20D4 :                    ;          Sector:     BYTE {R15}
4155/    20D4 :                    ;
4156/    20D4 :                    ;*****************************************************************
4157/    20D4 :
4158/    20D4 : D6 25 D4           Get_Cyl_H_S     call DivHdsSctrs        ; return Cylinder#, remainder = R0:3
4159/    20D7 : 70 E0                              push R0                 ; get ready to pass results to caller
4160/    20D9 : 70 E1                              push R1
4161/    20DB : D6 25 E0                           call DivSctrs           ; return with R1=Head, R2=Sector
4162/    20DE : F8 E3                              ld R15, R3              ; Sector
4163/    20E0 : E8 E2                              ld R14, R2              ; Head
4164/    20E2 : 50 ED                              pop R13                 ; LoCylinder
4165/    20E4 : 50 EC                              pop R12                 ; HiCylinder
4166/    20E6 : F9 22                              ld PSector, R15         ; update physical sector
4167/    20E8 : 8D 04 F8                           jp Bank_Ret
4168/    20EB :
4169/    20EB :
4170/    20EB :                    ;*****************************************************************
4171/    20EB :                    ;
4172/    20EB :                    ; Function: UpDate_Cur_Cyl
4173/    20EB :                    ;
4174/    20EB :                    ;  This function is responsible for reading the header off of
4175/    20EB :                    ;  an arbitrary sector. The cylinder information is extracted
4176/    20EB :                    ;  from the header and placed into the current cylinder memory
4177/    20EB :                    ;  locations.
4178/    20EB :                    ;
4179/    20EB :                    ; Inputs: none
4180/    20EB :                    ;
4181/    20EB :                    ; Outputs: UpDate_Cur_Cyl: BOOLEAN {zero flag is set if not(ReadHdr)}
4182/    20EB :                    ;
4183/    20EB :                    ; Global Variables Changed: Cur_Cylinder
4184/    20EB :                    ;
4185/    20EB :                    ; Algorithm:
4186/    20EB :                    ;
4187/    20EB :                    ;  Begin
4188/    20EB :                    ;    Temp:=GoodHdr(ReadHdr)
4189/    20EB :                    ;    If Temp
4190/    20EB :                    ;      Then Cur_Cylinder:=ReadHdr.Cylinder
4191/    20EB :                    ;    UpDate_Cur_Cyl:=Temp
4192/    20EB :                    ;  End
4193/    20EB :                    ;
4194/    20EB :                    ;*****************************************************************
4195/    20EB :
4196/    20EB : D6 20 FA           UpDate_Cur_Cyl  call GoodHdr
4197/    20EE : 6B 07                              jr Z, UD_C_C_End
4198/    20F0 : 09 50                              ld Cur_Cyl, R0
4199/    20F2 : 19 51                              ld Cur_Cyl+1, R1
4200/    20F4 : 46 E1 01                           or R1, #1               ; return non-zero status
4201/    20F7 : 8D 04 F8           UD_C_C_End      jp Bank_Ret
4202/    20FA :
4203/    20FA :
4204/    20FA :                    ;*****************************************************************
4205/    20FA :                    ;
4206/    20FA :                    ; Function: GoodHdr
4207/    20FA :                    ;
4208/    20FA :                    ;  This function is responsible for reading the header off of a
4209/    20FA :                    ;  sector and checking that header for validity, i.e. whether
4210/    20FA :                    ;  the cylinder, head, and sector (and their complements) look
4211/    20FA :                    ;  reasonable.
4212/    20FA :                    ;
4213/    20FA :                    ; Inputs: none
4214/    20FA :                    ;
4215/    20FA :                    ; Outputs: GoodHdr: BOOLEAN {zero flag is set if header invalid}
4216/    20FA :                    ;          GoodHdr.Cylinder: WORD {RR0}
4217/    20FA :                    ;          GoodHdr.Head:     BYTE {R2}
4218/    20FA :                    ;          GoodHdr.Sector:   BYTE {R3}
4219/    20FA :                    ;
4220/    20FA :                    ; Algorithm:
4221/    20FA :                    ;
4222/    20FA :                    ;  Begin
4223/    20FA :                    ;    ReadHdr
4224/    20FA :                    ;    If inverse(ReadHdr.Cylinder)=ReadHdr.InverseCylinder and
```

```
4225/    20FA :                    ;       inverse(ReadHdr.HdSctr)=ReadHdr.InverseHdSctr
4226/    20FA :                    ;     Then
4227/    20FA :                    ;       GoodHdr:=true
4228/    20FA :                    ;       GoodHdr.Cylinder:=ReadHdr.Cylinder
4229/    20FA :                    ;       GoodHdr.Head:=ReadHdr.HdSctr/bits 7:6
4230/    20FA :                    ;       GoodHdr.Sector:=ReadHdr.HdSctr/bits 5:0
4231/    20FA :                    ;     Else
4232/    20FA :                    ;       GoodHdr:=false
4233/    20FA :                    ;  End
4234/    20FA :                    ;
4235/    20FA :                    ;*****************************************************************
4236/    20FA :
4237/    20FA : 4C 08              GoodHdr         ld R4, #8               ; try hard to find a good header
4238/    20FC : D6 20 07           GdHdr_1         call ReadHdr
4239/    20FF : 2C 10                              ld R2, #RHHeader /256
4240/    2101 : 3C 0C                              ld R3, #RHHeader #256
4241/    2103 : 0C 40                              ld R0, #ScrReg0
4242/    2105 : 1C 06                              ld R1, #6               ; load 6 bytes
4243/    2107 : 83 02              GdHdr_Lp        ldei @R0, @RR2
4244/    2109 : 1A FC                              djnz R1, GdHdr_Lp
4245/    210B : 31 40                              srp #Wrk_Scr            ; context switch
4246/    210D :                           assume RP:Wrk_Scr
4247/    210D : B2 30                              xor R3, R0              ; check for valid header
4248/    210F : B2 41                              xor R4, R1
4249/    2111 : B2 52                              xor R5, R2
4250/    2113 : 52 34                              and R3, R4
4251/    2115 : 52 35                              and R3, R5
4252/    2117 : 60 E3                              com R3
4253/    2119 : 31 10                              srp #Wrk_Sys            ; context switch
4254/    211B :                           assume RP:Wrk_Sys
4255/    211B : B0 46                              clr ScrReg6             ; assume bad header
4256/    211D : 6B 13                              jr Z, GdHdr_2
4257/    211F : 76 24 80                           tm Excpt_Stat, #Recovery        ; check for Recovery on
4258/    2122 : 6B 25                              jr Z, GdHdr_End
4259/    2124 : D6 27 FC                           call ChkOff_NoOff       ; set auto-offset if not already on
4260/    2127 : 2C 00                              ld R2, #10 /256         ; wait 100ms before retrying
4261/    2129 : 3C 0A                              ld R3, #10 #256
4262/    212B : D6 01 CB                           call MsWait
4263/    212E : 4A CC                              djnz R4, GdHdr_1        ; retry if bad header
4264/    2130 : 8B 17                              jr GdHdr_End
4265/    2132 :                    ; found a good one
4266/    2132 : 08 40              GdHdr_2         ld R0, ScrReg0          ; GoodHdr.Cylinder:=ReadHdr.Cylinder
4267/    2134 : 18 41                              ld R1, ScrReg1
4268/    2136 : 28 42                              ld R2, ScrReg2          ; GoodHdr.Head:=ReadHdr.HdSctr/bits 7:6
4269/    2138 : F0 E2                              swap R2
4270/    213A : E0 E2                              rr R2
4271/    213C : E0 E2                              rr R2
4272/    213E : 56 E2 03                           and R2, #3              ; just leave head info in register
4273/    2141 : 38 42                              ld R3, ScrReg2          ; GoodHdr.Sectror:=ReadHdr.HdSctr/bits 5:0
4274/    2143 : 56 E3 3F                           and R3, #3Fh
4275/    2146 : E6 46 01                           ld ScrReg6, #1
4276/    2149 :                    ;
4277/    2149 : 44 46 46           GdHdr_End       or ScrReg6, ScrReg6     ; set zero flag
4278/    214C : 8D 04 F8                           jp Bank_Ret
4279/    214F :
4280/    214F :
4281/    214F :                    ;*****************************************************************
4282/    214F :                    ;
4283/    214F :                    ; Procedure: BlockMove
4284/    214F :                    ;
4285/    214F :                    ;  This procedure performs the following:
4286/    214F :                    ;
4287/    214F :                    ;  (Destination) <-- (Source)
4288/    214F :                    ;
4289/    214F :                    ;  where destination and source are both the same size (namely
4290/    214F :                    ;  exactly 532 bytes plus CRC and ECC).
4291/    214F :                    ;
4292/    214F :                    ; Inputs: Destination: PTR {RR2}
4293/    214F :                    ;         Source:      PTR {RR0}
4294/    214F :                    ;
4295/    214F :                    ; Outputs: none
4296/    214F :                    ;
4297/    214F :                    ;*****************************************************************
4298/    214F :
4299/    214F : D6 22 15           BlockMove       call Ext_Push
4300/    2152 : 4C 02                              ld R4, #BlockLength /256
4301/    2154 : 5C 1D                              ld R5, #BlockLength #256
4302/    2156 : 82 60              Blk_Move        lde R6, @RR0
4303/    2158 : 92 62                              lde @RR2, R6
4304/    215A : A0 E0                              incw RR0
4305/    215C : A0 E2                              incw RR2
4306/    215E : 80 E4                              decw RR4
4307/    2160 : EB F4                              jr NZ, Blk_Move
4308/    2162 : D6 22 36                           call Ext_Pop
4309/    2165 : AF                                 ret
4310/    2166 :
4311/    2166 :
4312/    2166 :                    ;*****************************************************************
4313/    2166 :                    ;
4314/    2166 :                    ; Function: UpDate_Hdr  {update header}
4315/    2166 :                    ;
4316/    2166 :                    ;  This function is responsible for comparing the header information
4317/    2166 :                    ;  in the buffer (it is assumed that a ReadHdr operation has just
4318/    2166 :                    ;  been performed and that the header info is currently residing in
4319/    2166 :                    ;  the ReadHdr buffer space) to that of the global cylinder variable.
4320/    2166 :                    ;  If the two do not match, then Cur_Cyl is updated to reflect the
4321/    2166 :                    ;  ReadHdr operation's findings and the function returns a False
4322/    2166 :                    ;  value indicating that a seek is in order.
4323/    2166 :                    ;
4324/    2166 :                    ; Inputs: none
4325/    2166 :                    ;
4326/    2166 :                    ; Outputs: UpDate_Hdr: BOOLEAN {zero flag is true if off-track}
4327/    2166 :                    ;
4328/    2166 :                    ; Global Variables Used: Cylinder
4329/    2166 :                    ;
4330/    2166 :                    ; Global Variables Changed: On_Track, Cur_Cyl
4331/    2166 :                    ;
4332/    2166 :                    ; Algorithm:
4333/    2166 :                    ;
4334/    2166 :                    ;  Begin
4335/    2166 :                    ;    DiskStatus.On_Track:=false
4336/    2166 :                    ;    If GoodHdr
4337/    2166 :                    ;      Then
4338/    2166 :                    ;       If RHHeader.Cylinder<>Cylinder
4339/    2166 :                    ;        Then
4340/    2166 :                    ;          Cur_Cyl:=RHHeader.Cylinder
4341/    2166 :                    ;          On_Track:=false
4342/    2166 :                    ;          UpDate_Hdr:=false
```

```
4343/    2166 :                     ;        Else
4344/    2166 :                     ;          UpDate_Hdr:=true
4345/    2166 :                     ;          DiskStatus.On_Track:=true
4346/    2166 :                     ;  End
4347/    2166 :                     ;
4348/    2166 :                     ;*****************************************************************
4349/    2166 :
4350/    2166 : D6 22 15            UpDate_Hdr      call Ext_Push
4351/    2169 : 56 56 7F                            and DiskStat, #0FFh-On_Track
4352/    216C : D6 20 FA                            call GoodHdr
4353/    216F : 6B 21                               jr Z, UpDt_Recal
4354/    2171 : 09 50               UpDate_OnTrck   ld Cur_Cyl, R0          ; Cur_Cyl:=RHHeader.Cylinder
4355/    2173 : 19 51                               ld Cur_Cyl+1, R1
4356/    2175 : B4 52 E0                            xor R0, Cylinder        ; check for correct cylinder address
4357/    2178 : B4 53 E1                            xor R1, Cylinder+1
4358/    217B : 42 01                               or R0, R1
4359/    217D : 0C 00                               ld R0, #0               ; assume failure
4360/    217F : EB 05                               jr NZ, UpDate_Err
4361/    2181 : 46 56 80                            or DiskStat, #On_Track
4362/    2184 : 0C 01                               ld R0, #1
4363/    2186 :                     ;
4364/    2186 : 42 00               UpDate_Err      or R0, R0
4365/    2188 : 70 FC               UpDate_H_End    push FLAGS
4366/    218A : D6 22 36                            call Ext_Pop
4367/    218D : 50 FC                               pop FLAGS
4368/    218F : 8D 04 F8                            jp Bank_Ret
4369/    2192 :                     ;
4370/    2192 : 0C 40               UpDt_Recal      ld R0, #DataRecal
4371/    2194 : D6 28 E2                            call Restore
4372/    2197 : 0C 00                               ld R0, #0
4373/    2199 : 8B EB                               jr UpDate_Err
4374/    219B :
4375/    219B :
4376/    219B :                     ;*****************************************************************
4377/    219B :                     ;
4378/    219B :                     ; Function: Get_Type
4379/    219B :                     ;
4380/    219B :                     ;  This function is responsible for converting ProFile-style
4381/    219B :                     ;  requests for the spare table and i.d. blocks into Widget
4382/    219B :                     ;  types and blocknumbers. Get_Type also performs a bounds check
4383/    219B :                     ;  on the blocknumber.
4384/    219B :                     ;
4385/    219B :                     ; Inputs: DriverType: BYTE {R8}
4386/    219B :                     ;
4387/    219B :                     ; Outputs: BlockType:   BYTE {R8}
4388/    219B :                     ;          BlockNumber: 3 BYTES {R12:14}
4389/    219B :                     ;
4390/    219B :                     ; Algorithm:
4391/    219B :                     ;
4392/    219B :                     ;  Begin
4393/    219B :                     ;    BlockNumber:=LogicalBlock
4394/    219B :                     ;    If DriverType=ProFile
4395/    219B :                     ;      Then
4396/    219B :                     ;        Case LogicalBlockNumber Of
4397/    219B :                     ;          FFFFFF: BlockType:=SpareTable
4398/    219B :                     ;              BlockNumber:=1
4399/    219B :                     ;          FFFFFE: BlockType:=ID
4400/    219B :                     ;              BlockNumber:=1
4401/    219B :                     ;        Otherwise: Type:=Data
4402/    219B :                     ;      Else Type:=Data
4403/    219B :                     ;    If BlockNumber>MaxLogicalBlockNumber
4404/    219B :                     ;      Then Abort
4405/    219B :                     ;  End
4406/    219B :                     ;
4407/    219B :                     ;*****************************************************************
4408/    219B :
4409/    219B : D6 06 0D            Get_Type        call Load_Logical
4410/    219E : 08 E8                               ld R0, R8
4411/    21A0 : 8C 02                               ld R8, #User_Type
4412/    21A2 : 42 00                               or R0, R0               ; If DriverType=ProFile
4413/    21A4 : EB 2C                               jr NZ, Get_Type_Check
4414/    21A6 : 0C FF                               ld R0, #0FFh            ; If BlockNumber=$-2...
4415/    21A8 : 1C FF                               ld R1, #0FFh
4416/    21AA : 2C FE                               ld R2, #0FEh
4417/    21AC : D6 03 A5                            call Sub3               ; compare
4418/    21AF : 42 01                               or R0, R1
4419/    21B1 : 42 02                               or R0, R2
4420/    21B3 : EB 0A                               jr NZ, G_T_ChkID
4421/    21B5 :                     ;
4422/    21B5 : 8C 08                               ld R8, #SprTbl_Type
4423/    21B7 : B0 EC               G_T_ProCnvrt    clr R12
4424/    21B9 : B0 ED                               clr R13
4425/    21BB : EC 01                               ld R14, #1
4426/    21BD : 8B 13                               jr Get_Type_Check
4427/    21BF :                     ;
4428/    21BF : 0C FF               G_T_ChkID       ld R0, #0FFh            ; If BlockNumber=$-1...
4429/    21C1 : 1C FF                               ld R1, #0FFh
4430/    21C3 : 2C FF                               ld R2, #0FFh
4431/    21C5 : D6 03 A5                            call Sub3               ; compare
4432/    21C8 : 42 01                               or R0, R1
4433/    21CA : 42 02                               or R0, R2
4434/    21CC : EB 04                               jr NZ, Get_Type_Check
4435/    21CE :                     ;
4436/    21CE : 8C 04                               ld R8, #ID_Type
4437/    21D0 : 8B E5                               jr G_T_ProCnvrt
4438/    21D2 :                     ;
4439/    21D2 : 0C 00               Get_Type_Check  ld R0, #HiMaxLogical    ; If BlockNumber>MaxLogical...
4440/    21D4 : 1C 4B                               ld R1, #MidMaxLogical
4441/    21D6 : 2C FF                               ld R2, #LoMaxLogical
4442/    21D8 : D6 03 A5                            call Sub3               ; compare
4443/    21DB : FB 0A                               jr NC, Get_Type_End
4444/    21DD :                     ;
4445/    21DD : 0C 02                               ld R0, #2
4446/    21DF : 1C 40                               ld R1, #Illegal_Block
4447/    21E1 : D6 04 03                            call SetStatus
4448/    21E4 : D6 05 1F                            call Abort
4449/    21E7 :                     ;
4450/    21E7 : 8D 04 F8            Get_Type_End    jp Bank_Ret
4451/    21EA :
4452/    21EA :
4453/    21EA :
4454/    21EA :                     ;*****************************************************************
4455/    21EA :                     ; Module Utils1.B1.Assem  {utilities, continued}
4456/    21EA :                     ;
4457/    21EA :                     ;       PROCEDURE Move4C(Source : PTR {RR2}
4458/    21EA :                     ;                        Destination : PTR {RR14})
4459/    21EA :                     ;       PROCEDURE Init_ExtStack
4460/    21EA :                     ;       PROCEDURE Ext_Push
```

```
4461/    21EA :                     ;       PROCEDURE Ext_Pop
4462/    21EA :                     ;       FUNCTION Load_TOS : PTR {RR2}
4463/    21EA :                     ;       PROCEDURE Store_TOS(Ptr : PTR {RR2})
4464/    21EA :                     ;       PROCEDURE ZeroBlock
4465/    21EA :                     ;
4466/    21EA :                     ;*********************************************************************
4467/    21EA :
4468/    21EA :                     ;*********************************************************************
4469/    21EA :                     ;
4470/    21EA :                     ; Procedure: Move4
4471/    21EA :                     ;
4472/    21EA :                     ;  This procedure is used to move 4 bytes from the source to the
4473/    21EA :                     ;  destination
4474/    21EA :                     ;
4475/    21EA :                     ; Inputs: Source:      PTR (RR2)
4476/    21EA :                     ;         Destination: PTR (RR14)
4477/    21EA :                     ;
4478/    21EA :                     ; Outputs: none
4479/    21EA :                     ;
4480/    21EA :                     ; Algorithm:
4481/    21EA :                     ;
4482/    21EA :                     ;  Begin
4483/    21EA :                     ;    For i:=0 To 3 Do
4484/    21EA :                     ;      @Destination[i]:=Source[i]
4485/    21EA :                     ;  End
4486/    21EA :                     ;
4487/    21EA :                     ;*********************************************************************
4488/    21EA :
4489/    21EA : 1C 04               Move4           ld R1, #4
4490/    21EC : 82 02               Move4_Lp        lde R0, @RR2
4491/    21EE : 92 0E                               lde @RR14, R0
4492/    21F0 : A0 E2                               incw RR2
4493/    21F2 : A0 EE                               incw RR14
4494/    21F4 : 1A F6                               djnz R1, Move4_Lp
4495/    21F6 : AF                                  ret
4496/    21F7 :
4497/    21F7 :
4498/    21F7 :                     ;*********************************************************************
4499/    21F7 :                     ;
4500/    21F7 :                     ; Procedure: Move4C  {move from program space to data space}
4501/    21F7 :                     ;
4502/    21F7 :                     ;  This procedure is used to move 4 bytes from the source to the
4503/    21F7 :                     ;  destination
4504/    21F7 :                     ;
4505/    21F7 :                     ; Inputs: Source:      PTR (RR2)
4506/    21F7 :                     ;         Destination: PTR (RR14)
4507/    21F7 :                     ;
4508/    21F7 :                     ; Outputs: none
4509/    21F7 :                     ;
4510/    21F7 :                     ; Algorithm:
4511/    21F7 :                     ;
4512/    21F7 :                     ;  Begin
4513/    21F7 :                     ;    For i:=0 To 3 Do
4514/    21F7 :                     ;      @Destination[i]:=Source[i]
4515/    21F7 :                     ;  End
4516/    21F7 :                     ;
4517/    21F7 :                     ;*********************************************************************
4518/    21F7 :
4519/    21F7 : 1C 04               Move4C          ld R1, #4
4520/    21F9 : C2 02               Move4C_Lp       ldc R0, @RR2
4521/    21FB : 92 0E                               lde @RR14, R0
4522/    21FD : A0 E2                               incw RR2
4523/    21FF : A0 EE                               incw RR14
4524/    2201 : 1A F6                               djnz R1, Move4C_Lp
4525/    2203 : AF                                  ret
4526/    2204 :
4527/    2204 :
4528/    2204 :                     ;*********************************************************************
4529/    2204 :                     ;
4530/    2204 :                     ; Procedure: Init_ExtStack
4531/    2204 :                     ;
4532/    2204 :                     ;  This procedure initializes Widget's external stack. This
4533/    2204 :                     ;  stack is a software implemented structire, and as such does not
4534/    2204 :                     ;  use the Z8's external stack capabilities
4535/    2204 :                     ;
4536/    2204 :                     ; Inputs: none
4537/    2204 :                     ;
4538/    2204 :                     ; Outputs: none
4539/    2204 :                     ;
4540/    2204 :                     ; Global Variables Changed: StackPtr
4541/    2204 :                     ;
4542/    2204 :                     ; Algorithm:
4543/    2204 :                     ;
4544/    2204 :                     ;  Begin
4545/    2204 :                     ;    StackPtr:=TopOfStack
4546/    2204 :                     ;  End
4547/    2204 :                     ;
4548/    2204 :                     ;*********************************************************************
4549/    2204 :
4550/    2204 : 2C 17               Init_ExtStack   ld R2, #StackPtr /256
4551/    2206 : 3C 4C                               ld R3, #StackPtr #256
4552/    2208 : 0C 17                               ld R0, #TopOfStack /256
4553/    220A : 92 02                               lde @RR2, R0
4554/    220C : A0 E2                               incw RR2
4555/    220E : 0C FF                               ld R0, #TopOfStack #256
4556/    2210 : 92 02                               lde @RR2, R0
4557/    2212 : 8D 04 F8                            jp Bank_Ret
4558/    2215 :
4559/    2215 :
4560/    2215 :                     ;*********************************************************************
4561/    2215 :                     ;
4562/    2215 :                     ; Procedure: Ext_Push
4563/    2215 :                     ;
4564/    2215 :                     ;  This procedure pushes the current working register set
4565/    2215 :                     ;  onto the external stack.
4566/    2215 :                     ;
4567/    2215 :                     ; Inputs: none
4568/    2215 :                     ;
4569/    2215 :                     ; Outputs: none
4570/    2215 :                     ;
4571/    2215 :                     ; Algorithm:
4572/    2215 :                     ;
4573/    2215 :                     ;  Begin
4574/    2215 :                     ;    For i:=0 To 15 Do
4575/    2215 :                     ;      @StackPtr:=WorkingRegister[15-i]
4576/    2215 :                     ;      StackPtr:=StackPtr-1
4577/    2215 :                     ;  End
4578/    2215 :                     ;
```

```
4579/    2215 :                     ;*********************************************************************
4580/    2215 :
4581/    2215 : E4 FD 44            Ext_Push        ld ScrReg4, RP          ; save context
4582/    2218 : 31 40                               srp #Wrk_Scr
4583/    221A :                             assume RP:Wrk_Scr
4584/    221A : D6 22 52                            call Load_TOS
4585/    221D : 08 E4                               ld R0, R4               ; save context
4586/    221F : 06 E0 0F                            add R0, #15             ; start at register 15
4587/    2222 : 1C 10                               ld R1, #16              ; save 16 registers
4588/    2224 : E3 F0               Ext_Push_Lp     ld R15, @R0
4589/    2226 : 92 F2                               lde @RR2, R15
4590/    2228 : 00 E0                               dec R0
4591/    222A : 80 E2                               decw RR2
4592/    222C : 1A F6                               djnz R1, Ext_Push_Lp
4593/    222E : D6 22 5D                            call Store_TOS
4594/    2231 : 49 FD                               ld RP, R4               ; restore context
4595/    2233 : 8D 04 F8                            jp Bank_Ret
4596/    2236 :
4597/    2236 :
4598/    2236 :                     ;*********************************************************************
4599/    2236 :                     ;
4600/    2236 :                     ; Procedure: Ext_Pop
4601/    2236 :                     ;
4602/    2236 :                     ;  This procedure pops 16 locations from the external stack
4603/    2236 :                     ;  into the current working register set.
4604/    2236 :                     ;
4605/    2236 :                     ; Inputs: none
4606/    2236 :                     ;
4607/    2236 :                     ; Outputs: none
4608/    2236 :                     ;
4609/    2236 :                     ; Algorithm:
4610/    2236 :                     ;
4611/    2236 :                     ;  Begin
4612/    2236 :                     ;    For i:=0 To 15 Do
4613/    2236 :                     ;      StackPtr:=StackPtr+1
4614/    2236 :                     ;      WorkingRegister[i]:=@StackPtr
4615/    2236 :                     ;  End
4616/    2236 :                     ;
4617/    2236 :                     ;*********************************************************************
4618/    2236 :
4619/    2236 : E4 FD 44            Ext_Pop         ld >ScrReg4, RP
4620/    2239 : 31 40                               srp #Wrk_Scr
4621/    223B :                             assume RP:Wrk_Scr
4622/    223B : D6 22 52                            call Load_TOS
4623/    223E : 08 E4                               ld R0, R4               ; save context
4624/    2240 : 1C 10                               ld R1, #16              ; load 16 registers
4625/    2242 : A0 E2                               incw RR2                ; StackPtr:=StackPtr+1
4626/    2244 : 83 02               Ext_Pop_Lp      ldei @R0, @RR2
4627/    2246 : 1A FC                               djnz R1, Ext_Pop_Lp
4628/    2248 : 80 E2                               decw RR2                ; point at TOS
4629/    224A : D6 22 5D                            call Store_TOS
4630/    224D : 49 FD                               ld RP, R4               ; restore context
4631/    224F : 8D 04 F8                            jp Bank_Ret
4632/    2252 :
4633/    2252 :
4634/    2252 :                     ;*********************************************************************
4635/    2252 :                     ;
4636/    2252 :                     ; Function: Load_TOS
4637/    2252 :                     ;
4638/    2252 :                     ;  This function returns a ptr to the current top
4639/    2252 :                     ;  of the external stack.
4640/    2252 :                     ;
4641/    2252 :                     ; Inputs: none
4642/    2252 :                     ;
4643/    2252 :                     ; Outputs: Load_TOS: PTR {RR2}
4644/    2252 :                     ;
4645/    2252 :                     ; Algorithm:
4646/    2252 :                     ;
4647/    2252 :                     ;  Begin
4648/    2252 :                     ;   Load_TOS:=StackPtr
4649/    2252 :                     ;  End
4650/    2252 :                     ;
4651/    2252 :                     ;*********************************************************************
4652/    2252 :
4653/    2252 : 0C 17               Load_TOS        ld      R0, #StackPtr /256
4654/    2254 : 1C 4C                               ld      R1, #StackPtr #256
4655/    2256 : 82 20                               lde     R2, @RR0
4656/    2258 : A0 E0                               incw    RR0
4657/    225A : 82 30                               lde     R3, @RR0
4658/    225C : AF                                  ret
4659/    225D :
4660/    225D :
4661/    225D :                     ;*********************************************************************
4662/    225D :                     ;
4663/    225D :                     ; Procedure: Store_TOS
4664/    225D :                     ;
4665/    225D :                     ;  This procedure stores the contents of Ptr into StackPtr.
4666/    225D :                     ;
4667/    225D :                     ; Inputs: Ptr: PTR {RR2}
4668/    225D :                     ;
4669/    225D :                     ; Outputs: none
4670/    225D :                     ;
4671/    225D :                     ; Algorithm:
4672/    225D :                     ;
4673/    225D :                     ;  Begin
4674/    225D :                     ;    StackPtr:=Ptr
4675/    225D :                     ;  End
4676/    225D :                     ;
4677/    225D :                     ;*********************************************************************
4678/    225D :
4679/    225D : A6 E3 4C            Store_TOS       cp R3, #StackPtr #256   ; check for stack underflow
4680/    2260 : BB 0D                               jr UGT, St_TOS
4681/    2262 :                     ;
4682/    2262 : 31 10                               srp #Wrk_Sys
4683/    2264 :                             assume RP:Wrk_Sys
4684/    2264 : 50 E4                               pop R4                  ; save some history of stack frame
4685/    2266 : 50 E5                               pop R5
4686/    2268 : 50 E6                               pop R6
4687/    226A : 50 E7                               pop R7
4688/    226C : D6 05 1F                            call Abort
4689/    226F :                     ;
4690/    226F : 0C 17               St_TOS          ld R0, #StackPtr /256
4691/    2271 : 1C 4C                               ld R1, #StackPtr #256
4692/    2273 : 92 20                               lde @RR0, R2
4693/    2275 : A0 E0                               incw RR0
4694/    2277 : 92 30                               lde @RR0, R3
4695/    2279 : AF                                  ret
4696/    227A :
```

```
4697/    227A :
4698/    227A :                     ;*********************************************************************
4699/    227A :                     ;
4700/    227A :                     ; Procedure: ZeroBlock
4701/    227A :                     ;
4702/    227A :                     ;  This procedure zeroes out the 532 bytes pointed to by the
4703/    227A :                     ;  input parameter.
4704/    227A :                     ;
4705/    227A :                     ; Inputs: BlockPtr: PTR {RR2}
4706/    227A :                     ;
4707/    227A :                     ; Outputs: none
4708/    227A :                     ;
4709/    227A :                     ;*********************************************************************
4710/    227A :
4711/    227A : EC 02               ZeroBlock       ld R14, #532 /256
4712/    227C : FC 14                               ld R15, #532 #256
4713/    227E : B0 E0                               clr R0
4714/    2280 : 92 02               ZeroBlk_Lp      lde @RR2, R0
4715/    2282 : A0 E2                               incw RR2
4716/    2284 : 80 EE                               decw RR14
4717/    2286 : EB F8                               jr NZ, ZeroBlk_Lp
4718/    2288 : 8D 04 F8                            jp Bank_Ret
4719/    228B :
4720/    228B :
4721/    228B :                     ;*********************************************************************
4722/    228B :                     ;
4723/    228B :                     ; Procedure: ZeroHeader
4724/    228B :                     ;
4725/    228B :                     ;  This procedure zeroes the sector gap, header sync gap,
4726/    228B :                     ;  header field, and data sync field
4727/    228B :                     ;
4728/    228B :                     ; Inputs: none
4729/    228B :                     ;
4730/    228B :                     ; Outputs: none
4731/    228B :                     ;
4732/    228B :                     ;*********************************************************************
4733/    228B :
4734/    228B : 1C 21               ZeroHeader      ld R1, #(WBuffer1 - WriteArray)
4735/    228D : 2C 10                               ld R2, #WriteArray /256
4736/    228F : 3C 00                               ld R3, #WriteArray #256
4737/    2291 : B0 E0                               clr R0
4738/    2293 : 92 02               ZeroHdr_Lp      lde @RR2, R0
4739/    2295 : A0 E2                               incw RR2
4740/    2297 : 1A FA                               djnz R1, ZeroHdr_Lp
4741/    2299 : 56 24 D7                            and Excpt_Stat, #0FFh-Nzero_Stat-Buf_Damage
4742/    229C : 8D 04 F8                            jp Bank_Ret
4743/    229F :
4744/    229F :
4745/    229F :
4746/    229F :                     ;*********************************************************************
4747/    229F :                     ; Module Spr1.B1.Assem
4748/    229F :                     ;
4749/    229F :                     ; This module contains those sparing routines that must
4750/    229F :                     ; be located in bank 1.
4751/    229F :                     ;
4752/    229F :                     ;       PROCEDURE Init_SprTbl
4753/    229F :                     ;       PROCEDURE Load_SprTbl
4754/    229F :                     ;       PROCEDURE SprChkSum
4755/    229F :                     ;       FUNCTION Chk_SprChk
4756/    229F :                     ;       FUNCTION Spr(SpareTableIndex : BYTE {R8}) : 3 BYTES {R12:14}
4757/    229F :                     ;       PROCEDURE UpDate_SprTbl
4758/    229F :                     ;       FUNCTION Chk_PassWord : BOOLEAN
4759/    229F :                     ;       PROCEDURE Load_PassWord(Destination : PTR {RR14})
4760/    229F :                     ;       PROCEDURE SpareCount( 2 BITS {R0/Bits 1:0} )
4761/    229F :                     ;
4762/    229F :                     ;*********************************************************************
4763/    229F :
4764/    229F :                     ;*********************************************************************
4765/    229F :                     ;
4766/    229F :                     ; Procedure: Init_SprTbl
4767/    229F :                     ;
4768/    229F :                     ;  This procedure initializes the spare table. It assumes
4769/    229F :                     ;  that there is NO spare table on disk.
4770/    229F :                     ;
4771/    229F :                     ; Inputs: FormatOffset:     BYTE (R4)
4772/    229F :                     ;         FormatInterLeave: BYTE (R5)
4773/    229F :                     ;
4774/    229F :                     ; Outputs: none
4775/    229F :                     ;
4776/    229F :                     ; Algorithm:
4777/    229F :                     ;
4778/    229F :                     ;  Begin
4779/    229F :                     ;    SparePw1:=$F0783C1E
4780/    229F :                     ;    SpareTmStmp:=0
4781/    229F :                     ;    For i:=0 To Length(SegPtrArray)-1 Do
4782/    229F :                     ;      SegPtrArray[i].Nil:=True
4783/    229F :                     ;      SegPtrArray[i].Ptr:=0
4784/    229F :                     ;    SpareCount:=0
4785/    229F :                     ;    BadCount:=0
4786/    229F :                     ;    For i:=0 To Length(SpareBitMap)-1 Do
4787/    229F :                     ;      SpareBitMap[i]:=0
4788/    229F :                     ;    For i:=0 To 1 Do
4789/    229F :                     ;      AddSpare(SprTbl_Type, Spare, GetNewSpare(Spr(i), Spr(i))
4790/    229F :                     ;    SparePw2:=$F0783C1E
4791/    229F :                     ;    UpDate_SprTbl
4792/    229F :                     ;  End
4793/    229F :                     ;
4794/    229F :                     ;*********************************************************************
4795/    229F :
4796/    229F : 2C 14               Init_SprTbl     ld R2, #SpareArray /256
4797/    22A1 : 3C BB                               ld R3, #SpareArray #256
4798/    22A3 : D6 22 7A                            call ZeroBlock
4799/    22A6 : EC 14                               ld R14, #SparePw1 /256
4800/    22A8 : FC BB                               ld R15, #SparePw1 #256
4801/    22AA : D6 25 06                            call Load_PassWord
4802/    22AD : 2C 14                               ld R2, #FmtOffset /256
4803/    22AF : 3C C3                               ld R3, #FmtOffset #256
4804/    22B1 : 92 42                               lde @RR2, R4            ; store offset value
4805/    22B3 : A0 E2                               incw RR2
4806/    22B5 : 92 52                               lde @RR2, R5            ; store interleave value
4807/    22B7 : A0 E2                               incw RR2
4808/    22B9 :                     ; initialize SegPtrArray
4809/    22B9 : 0C 80                               ld R0, #Nil
4810/    22BB : 1C 80                               ld R1, #SprCount-SegPtrArray
4811/    22BD : 92 02               I_S_Tbl_Lp2     lde @RR2, R0
4812/    22BF : A0 E2                               incw RR2
4813/    22C1 : 1A FA                               djnz R1, I_S_Tbl_Lp2
4814/    22C3 :                     ; initialize SpareBitMap
```

```
4815/    22C3 : 0C 00                              ld R0, #0
4816/    22C5 : 4C 01                              ld R4, #(SpareCheck-SprCount) /256
4817/    22C7 : 5C 4F                              ld R5, #(SpareCheck-SprCount) #256
4818/    22C9 : 92 02                 I_S_Tbl_Lp4  lde @RR2, R0
4819/    22CB : A0 E2                              incw RR2
4820/    22CD : 80 E4                              decw RR4
4821/    22CF : EB F8                              jr NZ, I_S_Tbl_Lp4
4822/    22D1 :                       ;
4823/    22D1 : E6 58 08                           ld Data_Type, #SprTbl_Type
4824/    22D4 : 5C 02                              ld R5, #2               ; create two tables
4825/    22D6 : B0 E8                              clr R8
4826/    22D8 : D6 24 29              Create_Tbl   call Spr
4827/    22DB : 2C 14                              ld R2, #GetNewSpare /256
4828/    22DD : 3C FB                              ld R3, #GetNewSpare #256
4829/    22DF : D6 04 AB                           call Bank_Call
4830/    22E2 : 70 E8                              push R8                 ; save counter
4831/    22E4 : F8 E0                              ld R15, R0
4832/    22E6 : 8C 08                              ld R8, #SprTbl_Type
4833/    22E8 : 46 E8 10                           or R8, #Spare
4834/    22EB : 2C 15                              ld R2, #AddSpare /256
4835/    22ED : 3C 68                              ld R3, #AddSpare #256
4836/    22EF : D6 04 AB                           call Bank_Call
4837/    22F2 : 50 E8                              pop R8
4838/    22F4 : 8E                                 inc R8                  ; go to next table
4839/    22F5 : 5A E1                              djnz R5, Create_Tbl
4840/    22F7 :                       ; create interleave mapping table
4841/    22F7 : 2C 16                              ld R2, #Map_Table /256
4842/    22F9 : 3C 81                              ld R3, #Map_Table #256
4843/    22FB : 1C 13                              ld R1, #NbrSctrs
4844/    22FD : 0C 00                              ld R0, #0
4845/    22FF : 92 02                 I_Map_Lp     lde @RR2, R0
4846/    2301 : A0 E2                              incw RR2
4847/    2303 : 06 E0 0C                           add R0, #Map_Dflt
4848/    2306 : A6 E0 13                           cp R0, #NbrSctrs
4849/    2309 : 1B 03                              jr LT, I_Map_Lp2
4850/    230B : 26 E0 13                           sub R0, #NbrSctrs
4851/    230E : 1A EF                 I_Map_Lp2    djnz R1, I_Map_Lp
4852/    2310 : EC 16                              ld R14, #SparePw2 /256
4853/    2312 : FC 96                              ld R15, #SparePw2 #256
4854/    2314 : D6 25 06                           call Load_PassWord
4855/    2317 :                       ;
4856/    2317 : D6 24 3F                           call UpDate_SprTbl
4857/    231A : 8D 04 F8                           jp Bank_Ret
4858/    231D :
4859/    231D :
4860/    231D :                       ;*********************************************************************
4861/    231D :                       ;
4862/    231D :                       ; Procedure: Load_SprTbl
4863/    231D :                       ;
4864/    231D :                       ;  This procedure loads the spare table from disk. The table is
4865/    231D :                       ;  found by a linear search of all spare blocks until a block
4866/    231D :                       ;  is found where both the Spare Table Identifier field is present
4867/    231D :                       ;  and the internal passwords and check byte are valid.
4868/    231D :                       ;
4869/    231D :                       ; Inputs: none
4870/    231D :                       ;
4871/    231D :                       ; Outputs: none
4872/    231D :                       ;
4873/    231D :                       ; Algorithm:
4874/    231D :                       ;
4875/    231D :                       ;  Begin
4876/    231D :                       ;    Seek_Type:=Access_Offset
4877/    231D :                       ;    InterLeaveFactor:=Find_InterLeave
4878/    231D :                       ;    Found:=false
4879/    231D :                       ;    Count:=0
4880/    231D :                       ;    i:=NumberOfSpareBlocks
4881/    231D :                       ;    While i>0 and Count<2 Do
4882/    231D :                       ;      Seek(CnvrtLogical(MulR0_m(i)))
4883/    231D :                       ;      j:=0
4884/    231D :                       ;      Sector:=0
4885/    231D :                       ;      While not(Found) And (j<NbrSctrs) Do
4886/    231D :                       ;        If ReadCommon
4887/    231D :                       ;          Then
4888/    231D :                       ;            If BlockID=SprTblID and
4889/    231D :                       ;                 PassWord1=PassWord2=PassWord and
4890/    231D :                       ;                 CheckByte is valid
4891/    231D :                       ;             Then
4892/    231D :                       ;               Found:=true
4893/    231D :                       ;               Count:=Count+1
4894/    231D :                       ;               If Count>1
4895/    231D :                       ;                 Then
4896/    231D :                       ;                   If SpareTable.RunNumber<ReadBuffer.RunNumber
4897/    231D :                       ;                     Then MoveBlock(SpareTable, ReadBuffer)
4898/    231D :                       ;                     Else MoveBlock(SpareTable, ReadBuffer)
4899/    231D :                       ;          Else
4900/    231D :                       ;            If RdErrCnt<10
4901/    231D :                       ;              Then
4902/    231D :                       ;                MoveBlock(ReadBuffer, Buffer2)
4903/    231D :                       ;                If BlockID=SprTblID and
4904/    231D :                       ;                     PassWord1=PassWord2=PassWord and
4905/    231D :                       ;                     CheckByte is valid
4906/    231D :                       ;                  Then
4907/    231D :                       ;                    Found:=true
4908/    231D :                       ;                    Count:=Count+1
4909/    231D :                       ;                    If Count>1
4910/    231D :                       ;                      Then
4911/    231D :                       ;                        If SpareTable.RunNumber<ReadBuffer.RunNumber
4912/    231D :                       ;                          Then MoveBlock(SpareTable, ReadBuffer)
4913/    231D :                       ;                        Else MoveBlock(SpareTable, ReadBuffer)
4914/    231D :                       ;        j:=j+1
4915/    231D :                       ;        Sector:=(Secotr+InterLeaveFactor) mod NbrSctrs
4916/    231D :                       ;      i:=i+1
4917/    231D :                       ;    If not(Found)
4918/    231D :                       ;      Then Abort
4919/    231D :                       ;      Else
4920/    231D :                       ;        UpDate_SprTbl
4921/    231D :                       ;        Park
4922/    231D :                       ;        SlfTst_Result.NoSpareTable:=False
4923/    231D :                       ;  End
4924/    231D :                       ;
4925/    231D :                       ;*********************************************************************
4926/    231D :
4927/    231D : 5C 4C                 Load_SprTbl  ld R5, #76              ; i:= NumberOfSpareBlocks
4928/    231F : B0 E4                              clr R4                  ; SprTbl_Found:=false
4929/    2321 : E6 58 02                           ld Data_Type, #User_Type
4930/    2324 : D6 22 15              L_SprTbl_Lp  call Ext_Push
4931/    2327 : 08 E5                              ld R0, R5
4932/    2329 : D6 25 CB                           call MulR0_m
```

```
4933/    232C : D6 20 D4                          call Get_Cyl_H_S
4934/    232F : E6 57 80                          ld Seek_Type, #Access
4935/    2332 : D6 26 05                          call Seek
4936/    2335 : D6 22 36                          call Ext_Pop
4937/    2338 : 6C 13                             ld R6, #NbrSctrs        ; check the entire track
4938/    233A : B0 55                             clr Sector              ; starting with sector 0
4939/    233C :                          ;
4940/    233C : 70 E4                    L_Rd_Lp  push R4
4941/    233E : 70 E5                             push R5
4942/    2340 : 70 E6                             push R6
4943/    2342 : 2C 10                             ld R2, #RdBlk_Vector /256
4944/    2344 : 3C 83                             ld R3, #RdBlk_Vector #256
4945/    2346 : D6 04 AB                          call Bank_Call
4946/    2349 : 50 E6                             pop R6
4947/    234B : 50 E5                             pop R5
4948/    234D : 50 E4                             pop R4
4949/    234F : EB 1C                             jr NZ, Chk_SprTbl
4950/    2351 :                          ;
4951/    2351 : 08 27                             ld R0, RdErrCnt
4952/    2353 : 56 E0 0F                          and R0, #0Fh            ; mask unwanted status
4953/    2356 : A6 E0 0A                          cp R0, #10              ; check for any successful reads
4954/    2359 : 9B 05                             jr GE, L_SprTbl_More
4955/    235B : D6 20 5B                          call Buf2_To_RBuf       ; get good data back into read buffer
4956/    235E : 8B 0D                             jr Chk_SprTbl
4957/    2360 :                          ;
4958/    2360 : 20 55                    L_SprTbl_More   inc Sector
4959/    2362 : 6A D8                             djnz R6, L_Rd_Lp
4960/    2364 : 5A BE                             djnz R5, L_SprTbl_Lp
4961/    2366 : 42 44                             or R4, R4               ; check if any spare table found
4962/    2368 : EB 62                             jr NZ, L_Spr_End
4963/    236A : D6 05 1F                          call Abort
4964/    236D :                          ;
4965/    236D : E6 42 12                 Chk_SprTbl      ld ScrReg2, #(RBuffer1+BlockID) /256
4966/    2370 : E6 43 19                          ld ScrReg3, #(RBuffer1+BlockID) #256
4967/    2373 : D6 24 E5                          call Chk_PassWord
4968/    2376 : 6B E8                             jr Z, L_SprTbl_More
4969/    2378 :                          ;
4970/    2378 : E6 42 10                          ld ScrReg2, #RBuffer1 /256
4971/    237B : E6 43 19                          ld ScrReg3, #RBuffer1 #256
4972/    237E : D6 24 E5                          call Chk_PassWord
4973/    2381 : 6B DD                             jr Z, L_SprTbl_More
4974/    2383 :                          ;
4975/    2383 : 2C 11                             ld R2, #(RBuffer1+SpareCheck-SpareArray) /256
4976/    2385 : 3C F2                             ld R3, #(RBuffer1+SpareCheck-SpareArray) #256
4977/    2387 : 82 02                             lde R0, @RR2            ; check possible check byte
4978/    2389 : A0 E2                             incw RR2
4979/    238B : 82 12                             lde R1, @RR2
4980/    238D : 31 40                             srp #Wrk_Scr
4981/    238F :                                  assume RP:Wrk_Scr
4982/    238F : CC 10                             ld R12, #RBuffer1 /256
4983/    2391 : DC 19                             ld R13, #RBuffer1 #256
4984/    2393 : D6 23 E7                          call SprChk2
4985/    2396 : 31 10                             srp #Wrk_Sys
4986/    2398 :                                  assume RP:Wrk_Sys
4987/    2398 : D6 24 16                          call Chk_Spr2
4988/    239B : 6B C3                             jr Z, L_SprTbl_More
4989/    239D :                          ;
4990/    239D : 42 44                             or R4, R4               ; check for a SpareTable already found
4991/    239F : 6B 22                             jr Z, L_Spr_Move
4992/    23A1 : 31 40                             srp #Wrk_Scr
4993/    23A3 :                                  assume RP:Wrk_Scr
4994/    23A3 : 6C 14                             ld R6, #SpareTmStmp /256
4995/    23A5 : 7C BF                             ld R7, #SpareTmStmp #256
4996/    23A7 : 4C 40                             ld R4, #ScrReg0
4997/    23A9 : D6 23 DC                          call Ld_TmStmp
4998/    23AC : 6C 10                             ld R6, #(RBuffer1+SpareTmStmp-SpareArray) /256
4999/    23AE : 7C 1D                             ld R7, #(RBuffer1+SpareTmStmp-SpareArray) #256
5000/    23B0 : 4C 4C                             ld R4, #ScrRegC
5001/    23B2 : D6 23 DC                          call Ld_TmStmp
5002/    23B5 : 22 3F                             sub R3, R15
5003/    23B7 : 32 2E                             sbc R2, R14
5004/    23B9 : 32 1D                             sbc R1, R13
5005/    23BB : 32 0C                             sbc R0, R12
5006/    23BD : 31 10                             srp #Wrk_Sys
5007/    23BF :                                  assume RP:Wrk_Sys
5008/    23BF : 9B 05                             jr GE, L_Spr_Inc
5009/    23C1 : 00 E4                             dec R4                  ; account for old, bogus spare table
5010/    23C3 : D6 20 89                 L_Spr_Move      call RBuf_To_Spr
5011/    23C6 : 4E                       L_Spr_Inc       inc R4                  ; note the arrval of a SpareTable
5012/    23C7 : A6 E4 02                          cp R4, #2               ; see if that's all there is
5013/    23CA : EB 94                             jr NZ, L_SprTbl_More
5014/    23CC :                          ;
5015/    23CC : D6 24 3F                 L_Spr_End       call UpDate_SprTbl
5016/    23CF : 56 25 FE                          and SlfTst_Result, #0FFh-No_SprTbl
5017/    23D2 : 2C 29                             ld R2, #Park_Heads /256
5018/    23D4 : 3C 88                             ld R3, #Park_Heads #256
5019/    23D6 : D6 04 AB                          call Bank_Call
5020/    23D9 : 8D 04 F8                          jp Bank_Ret
5021/    23DC :                          ;*****************************************************************
5022/    23DC : 5C 04                    Ld_TmStmp       ld R5, #4               ; load 4 bytes
5023/    23DE : 83 46                    Ld_Tm_Lp        ldei @R4, @RR6
5024/    23E0 : 5A FC                             djnz R5, Ld_Tm_Lp
5025/    23E2 : AF                                ret
5026/    23E3 :
5027/    23E3 :
5028/    23E3 :                          ;*****************************************************************
5029/    23E3 :                          ;
5030/    23E3 :                          ; Procedure: SprChkSum
5031/    23E3 :                          ;
5032/    23E3 :                          ;  This procedure calculates a 16-bit checksum over the contents
5033/    23E3 :                          ;  of the spare table, and stores the sum within the spare table.
5034/    23E3 :                          ;
5035/    23E3 :                          ; Inputs: none
5036/    23E3 :                          ;
5037/    23E3 :                          ; Outputs: none
5038/    23E3 :                          ;
5039/    23E3 :                          ; Side Effect: ScrReg1, ScrReg2 hold the calculated check byte on return
5040/    23E3 :                          ;
5041/    23E3 :                          ; Algorithm:
5042/    23E3 :                          ;
5043/    23E3 :                          ;  Begin
5044/    23E3 :                          ;    Sum:=0
5045/    23E3 :                          ;    SumPtr:=SpareArray
5046/    23E3 :                          ;    For i:=1 To Length(SpareArray) Do
5047/    23E3 :                          ;      Sum:=Sum+SpareArray[i-1]
5048/    23E3 :                          ;    SpareArray.ChkSum:=Sum
5049/    23E3 :                          ;  End
5050/    23E3 :                          ;
```

```
5051/    23E3 :                     ;*********************************************************************
5052/    23E3 :
5053/    23E3 : CC 14               SprChkSum       ld R12, #SpareArray /256
5054/    23E5 : DC BB                               ld R13, #SpareArray #256
5055/    23E7 : EC 01               SprChk2         ld R14, #(SpareCheck-SpareArray) /256
5056/    23E9 : FC D9                               ld R15, #(SpareCheck-SpareArray) #256
5057/    23EB : B0 E1                               clr R1
5058/    23ED : B0 E2                               clr R2
5059/    23EF : 82 0C               SprChk_Lp       lde R0, @RR12
5060/    23F1 : 02 20                               add R2, R0
5061/    23F3 : 16 E1 00                            adc R1,#0
5062/    23F6 : A0 EC                               incw RR12
5063/    23F8 : 80 EE                               decw RR14
5064/    23FA : EB F3                               jr NZ, SprChk_Lp
5065/    23FC : 92 1C                               lde @RR12, R1           ; store high byte of checksum
5066/    23FE : A0 EC                               incw RR12
5067/    2400 : 92 2C                               lde @RR12, R2           ; store low byte of checksum
5068/    2402 : 8D 04 F8                            jp Bank_Ret
5069/    2405 :
5070/    2405 :
5071/    2405 :                     ;*********************************************************************
5072/    2405 :                     ;
5073/    2405 :                     ; Function: Chk_SprChk
5074/    2405 :                     ;
5075/    2405 :                     ;  This function is responsible for verifying that the checksum
5076/    2405 :                     ;  residing in the spare table is correct.
5077/    2405 :                     ;
5078/    2405 :                     ; Inputs: none
5079/    2405 :                     ;
5080/    2405 :                     ; Outputs: Chk_SprChk: BOOLEAN {zero flag}
5081/    2405 :                     ;
5082/    2405 :                     ; Algorithm:
5083/    2405 :                     ;
5084/    2405 :                     ;  Begin
5085/    2405 :                     ;    TempSum:=SpareTable.CheckSum
5086/    2405 :                     ;    SpareTable.CheckSum:=SprChkSum
5087/    2405 :                     ;    If TempSum=SpareTable.CheckSum
5088/    2405 :                     ;      Then Chk_SprChk:=true
5089/    2405 :                     ;      Else Chk_SprChk:=false
5090/    2405 :                     ;  End
5091/    2405 :                     ;
5092/    2405 :                     ;*********************************************************************
5093/    2405 :
5094/    2405 : 2C 16               Chk_SprChk      ld R2, #SpareCheck /256
5095/    2407 : 3C 94                               ld R3, #SpareCheck #256
5096/    2409 : 82 02                               lde R0, @RR2
5097/    240B : A0 E2                               incw RR2
5098/    240D : 82 12                               lde R1, @RR2
5099/    240F : 31 40                               srp #Wrk_Scr
5100/    2411 :                             assume RP:Wrk_Scr
5101/    2411 : D6 23 E3                            call SprChkSum
5102/    2414 : 31 10                               srp #Wrk_Sys
5103/    2416 :                             assume RP:Wrk_Sys
5104/    2416 : B4 41 E0            Chk_Spr2        xor R0, ScrReg1         ; side effect: ScrReg 1:2 hold new checkbyte
5105/    2419 : B4 42 E1                            xor R1, ScrReg2
5106/    241C : 42 01                               or R0, R1
5107/    241E : B0 E0                               clr R0
5108/    2420 : EB 02                               jr NZ, Chk_Spr_End
5109/    2422 : 0C 01                               ld R0, #1
5110/    2424 : 42 00               Chk_Spr_End     or R0, R0               ; set zero flag
5111/    2426 : 8D 04 F8                            jp Bank_Ret
5112/    2429 :
5113/    2429 :
5114/    2429 :                     ;*********************************************************************
5115/    2429 :                     ;
5116/    2429 :                     ; Function: Spr
5117/    2429 :                     ;
5118/    2429 :                     ;  This function returns the logical block number associated
5119/    2429 :                     ;  with the parameter passed in. Because there are only two spare
5120/    2429 :                     ;  blocks containing the spare table, this function accepts
5121/    2429 :                     ;  only ODD or EVEN input params.
5122/    2429 :                     ;
5123/    2429 :                     ; Inputs: SpareTableIndex: BYTE {R8}
5124/    2429 :                     ;
5125/    2429 :                     ; Outputs: Spr: 3 BYTES {R12:14}
5126/    2429 :                     ;
5127/    2429 :                     ; Algorithm:
5128/    2429 :                     ;
5129/    2429 :                     ;  Begin
5130/    2429 :                     ;    If SpareTableIndex is EVEN
5131/    2429 :                     ;      Then Spr:=SprBlk0
5132/    2429 :                     ;      Else Spr:=SprBlk1
5133/    2429 :                     ;  End
5134/    2429 :                     ;
5135/    2429 :                     ;*********************************************************************
5136/    2429 :
5137/    2429 : 56 E8 01            Spr             and R8, #1              ; If SpareTableIndex is EVEN
5138/    242C : EB 08                               jr NZ, Spr_Odd
5139/    242E :                     ;
5140/    242E : CC 00                               ld R12, #HiSpr0
5141/    2430 : DC 19                               ld R13, #MidSpr0
5142/    2432 : EC 55                               ld R14, #LoSpr0
5143/    2434 : 8B 06                               jr Spr_End
5144/    2436 :                     ;
5145/    2436 : CC 00               Spr_Odd         ld R12, #HiSpr1
5146/    2438 : DC 32                               ld R13, #MidSpr1
5147/    243A : EC AA                               ld R14, #LoSpr1
5148/    243C : 8D 04 F8            Spr_End         jp Bank_Ret
5149/    243F :
5150/    243F :
5151/    243F :                     ;*********************************************************************
5152/    243F :                     ;
5153/    243F :                     ; Procedure: UpDate_SprTbl
5154/    243F :                     ;
5155/    243F :                     ;  This procedure is responsible for updating the spare table
5156/    243F :                     ;  to both of its locations on disk after a change has been made
5157/    243F :                     ;  to the table
5158/    243F :                     ;
5159/    243F :                     ; Inputs: none
5160/    243F :                     ;
5161/    243F :                     ; Outputs: none
5162/    243F :                     ;
5163/    243F :                     ; Algorithm:
5164/    243F :                     ;
5165/    243F :                     ;  Begin
5166/    243F :                     ;    SpareTmStamp:=SpareTmStamp+1
5167/    243F :                     ;    SprChkSum
5168/    243F :                     ;    ZeroBlock
```

```
5169/    243F :                    ;    WBuffer1.BlockID:=PassWord
5170/    243F :                    ;    For i:=0 To 1 Do
5171/    243F :                    ;      MoveBlock(WriteBuffer, SpareTable)
5172/    243F :                    ;      TempCyl, TempHead, TempSector:=
5173/    243F :                    ;                    Get_Cyl_H_S(SrchSpTabl(SpareTable(Spr(i)))
5174/    243F :                    ;      If table not found in SpareTable Then Abort
5175/    243F :                    ;      Seek(TemoCyl, TempHead, TempSector)
5176/    243F :                    ;      If nor(WriteVerify(Conservative))
5177/    243F :                    ;        Then
5178/    243F :                    ;          If Recovery And WriteVerify.ErrorCode=Ex_ReadErr
5179/    243F :                    ;            Then Exit UpDate_SprTbl
5180/    243F :                    ;            Else
5181/    243F :                    ;              ZeroBlock
5182/    243F :                    ;              WriteBlock
5183/    243F :                    ;              MoveBlock(Buffer2, SpareTable)
5184/    243F :                    ;              SpareBlock(True, SpareTable, Write_Op, Spr(i))
5185/    243F :                    ;  End
5186/    243F :                    ;
5187/    243F :                    ;*****************************************************************
5188/    243F :
5189/    243F : C8 52              UpDate_SprTbl   ld R12, Cylinder        ; save current seek address
5190/    2441 : D8 53                              ld R13, Cylinder+1
5191/    2443 : E8 54                              ld R14, Head
5192/    2445 : F8 55                              ld R15, Sector
5193/    2447 : D6 22 15                           call Ext_Push
5194/    244A :                    ;
5195/    244A : 2C 14                              ld R2, #SpareTmStmp /256
5196/    244C : 3C BF                              ld R3, #SpareTmStmp #256
5197/    244E : 0C 1C                              ld R0, #Wrk_Sys+12
5198/    2450 : 1C 04                              ld R1, #4               ; get 4 bytes
5199/    2452 : 83 02              UpDate_1        ldei @R0, @RR2
5200/    2454 : 1A FC                              djnz R1, UpDate_1
5201/    2456 : 06 EF 01                           add R15, #1             ; increment the count
5202/    2459 : 16 EE 00                           adc R14, #0
5203/    245C : 16 ED 00                           adc R13, #0
5204/    245F : 16 EC 00                           adc R12, #0
5205/    2462 : 0C 1C                              ld R0, #Wrk_Sys+12
5206/    2464 : 1C 04                              ld R1, #4               ; move 4 bytes
5207/    2466 : 2C 14                              ld R2, #SpareTmStmp /256
5208/    2468 : 3C BF                              ld R3, #SpareTmStmp #256
5209/    246A : 93 02              UpDate_2        ldei @RR2, @R0
5210/    246C : 1A FC                              djnz R1, UpDate_2
5211/    246E : D6 23 E3                           call SprChkSum
5212/    2471 : 4C 02                              ld R4, #2               ; write the table to the disk twice
5213/    2473 : B0 E5                              clr R5                  ; spare table index
5214/    2475 : D6 20 AC           SprB_Lp         call Spr_To_WrBuf
5215/    2478 : EC 12                              ld R14, #(WBuffer1+BlockID) /256
5216/    247A : FC 21                              ld R15, #(WBuffer1+BlockID) #256
5217/    247C : D6 25 06                           call Load_PassWord
5218/    247F : 88 E5                              ld R8, R5
5219/    2481 : D6 24 29                           call Spr
5220/    2484 : E6 58 08                           ld Data_Type, #SprTbl_Type
5221/    2487 : 2C 16                              ld R2, #CnvrtLogical /256
5222/    2489 : 3C 3A                              ld R3, #CnvrtLogical #256
5223/    248B : D6 04 AB                           call Bank_Call
5224/    248E : EB 03                              jr NZ, SprB_Seek
5225/    2490 : D6 05 1F                           call Abort
5226/    2493 :                    ;
5227/    2493 : E6 57 90           SprB_Seek       ld Seek_Type, #Access_Offset
5228/    2496 : D6 26 05                           call Seek
5229/    2499 : E6 5A 01                           ld BlkStat, #S_Block            ; say that we've got a spare block
5230/    249C : E6 58 08                           ld Data_Type, #SprTbl_Type      ; ditto
5231/    249F : E6 27 0A                           ld RdErrCnt, #10                ; prime the counter for write error
5232/    24A2 : 2C 19                              ld R2, #WrVer_Common /256
5233/    24A4 : 3C 64                              ld R3, #WrVer_Common #256
5234/    24A6 : D6 04 AB                           call Bank_Call
5235/    24A9 : EB 2E                              jr NZ, Spr_Next
5236/    24AB :                    ;
5237/    24AB : 76 24 80                           tm Excpt_Stat, #Recovery        ; If Recovery Then ...
5238/    24AE : EB 03                              jr NZ, Rcvr_SprTbl
5239/    24B0 : D6 05 1F                           call Abort
5240/    24B3 :                    ;
5241/    24B3 : 18 27              Rcvr_SprTbl     ld R1, RdErrCnt         ; check for noisy read
5242/    24B5 : 56 E1 0F                           and R1, #0Fh
5243/    24B8 : A6 E1 03                           cp R1, #SprThresh
5244/    24BB : 2B 1C                              jr LE, Spr_Next
5245/    24BD : 2C 10                              ld R2, #WBuffer1 /256
5246/    24BF : 3C 21                              ld R3, #WBuffer1 #256
5247/    24C1 : D6 22 7A                           call ZeroBlock
5248/    24C4 : 2C 18                              ld R2, #Wr_Common /256
5249/    24C6 : 3C CF                              ld R3, #Wr_Common #256
5250/    24C8 : D6 04 AB                           call Bank_Call
5251/    24CB : 88 E5                              ld R8, R5
5252/    24CD : D6 24 29                           call Spr
5253/    24D0 : 2C 13                              ld R2, #Spr_Enter /256
5254/    24D2 : 3C EA                              ld R3, #Spr_Enter #256
5255/    24D4 : D6 04 AB                           call Bank_Call
5256/    24D7 : 6B DA                              jr Z, Rcvr_SprTbl
5257/    24D9 :                    ;
5258/    24D9 : 5E                 Spr_Next        inc R5                  ; do next table
5259/    24DA : 4A 99                              djnz R4, SprB_Lp
5260/    24DC : D6 22 36                           call Ext_Pop
5261/    24DF : D6 26 05                           call Seek               ; get back to original seek address
5262/    24E2 : 8D 04 F8                           jp Bank_Ret
5263/    24E5 :
5264/    24E5 :
5265/    24E5 :                    ;*****************************************************************
5266/    24E5 :                    ;
5267/    24E5 :                    ; Function: Chk_PassWord
5268/    24E5 :                    ;
5269/    24E5 :                    ;  This function is responsible for checking if the 32 bits
5270/    24E5 :                    ;  pointed to by the input parameter match with the controller's
5271/    24E5 :                    ;  32 bit password
5272/    24E5 :                    ;
5273/    24E5 :                    ; Inputs: PassWordPtr: PTR {ScrReg2:3}
5274/    24E5 :                    ;
5275/    24E5 :                    ; Outputs: Chk_PassWord: BOOLEAN {zero flag}
5276/    24E5 :                    ;
5277/    24E5 :                    ; Algorithm:
5278/    24E5 :                    ;
5279/    24E5 :                    ;  Begin
5280/    24E5 :                    ;    If @PassWordPtr=PassWord
5281/    24E5 :                    ;      Then Chk_PassWord:=true
5282/    24E5 :                    ;      Else Chk_PassWord:=false
5283/    24E5 :                    ;  End
5284/    24E5 :                    ;
5285/    24E5 :                    ;*****************************************************************
5286/    24E5 :
```

```
5287/    24E5 : 31 40              Chk_PassWord    srp #Wrk_Scr
5288/    24E7 :                            assume RP:Wrk_Scr
5289/    24E7 : 4C 04                      ld R4, #4               ; check 4 bytes
5290/    24E9 : EC 10                      ld R14, #PassWord /256
5291/    24EB : FC 03                      ld R15, #PassWord #256
5292/    24ED : C2 0E              Chk_P_Lp        ldc R0, @RR14           ; get a byte of the password
5293/    24EF : A0 EE                      incw RR14
5294/    24F1 : 82 12                      lde R1, @RR2            ; get a byte of test string
5295/    24F3 : A0 E2                      incw RR2
5296/    24F5 : A2 01                      cp R0, R1
5297/    24F7 : B0 E0                      clr R0                  ; assume failure
5298/    24F9 : EB 04                      jr NZ, Chk_P_End
5299/    24FB : 4A F0                      djnz R4, Chk_P_Lp
5300/    24FD : 0C 01                      ld R0, #1
5301/    24FF : 42 00              Chk_P_End       or R0, R0               ; set flags
5302/    2501 : 31 10                      srp #Wrk_Sys
5303/    2503 :                            assume RP:Wrk_Sys
5304/    2503 : 8D 04 F8                   jp Bank_Ret
5305/    2506 :
5306/    2506 :
5307/    2506 :                    ;*****************************************************************
5308/    2506 :                    ;
5309/    2506 :                    ; Procedure: Load_PassWord
5310/    2506 :                    ;
5311/    2506 :                    ;  This procedure loads the 32-bit password into the memory
5312/    2506 :                    ;  location pointed to by the input parameter
5313/    2506 :                    ;
5314/    2506 :                    ; Inputs: Destination: PTR {RR14}
5315/    2506 :                    ;
5316/    2506 :                    ; Outputs: none
5317/    2506 :                    ;
5318/    2506 :                    ;*****************************************************************
5319/    2506 :
5320/    2506 : 2C 10              Load_PassWord   ld R2, #PassWord /256
5321/    2508 : 3C 03                      ld R3, #PassWord #256
5322/    250A : 1C 04                      ld R1, #4               ; move 4 bytes
5323/    250C : C2 02              Lpw_Lp          ldc R0, @RR2
5324/    250E : 92 0E                      lde @RR14, R0
5325/    2510 : A0 E2                      incw RR2
5326/    2512 : A0 EE                      incw RR14
5327/    2514 : 1A F6                      djnz R1, Lpw_Lp
5328/    2516 : 8D 04 F8                   jp Bank_Ret
5329/    2519 :
5330/    2519 :
5331/    2519 :                    ;*****************************************************************
5332/    2519 :                    ;
5333/    2519 :                    ; Procedure: SpareCount
5334/    2519 :                    ;
5335/    2519 :                    ;  This procedure is responsible for incrementing/decrementing
5336/    2519 :                    ;  the Spare/Bad Block count. If the total count exceeds
5337/    2519 :                    ;  MaxSpares-5 then a status bit is set for this command only.
5338/    2519 :                    ;
5339/    2519 :                    ; Inputs: Command: 2 BITS {R0/Bits 1:0}
5340/    2519 :                    ;
5341/    2519 :                    ; Outputs: none
5342/    2519 :                    ;
5343/    2519 :                    ; Algorithm:
5344/    2519 :                    ;
5345/    2519 :                    ;  Begin
5346/    2519 :                    ;    Case Command Of
5347/    2519 :                    ;      0: Increment the spare count
5348/    2519 :                    ;      1: Increment the bad block count
5349/    2519 :                    ;      2: Decrement the bad block count
5350/    2519 :                    ;      Otherwise Abort
5351/    2519 :                    ;    If SpareCount+BadBlockCount>=MaxSpares-5
5352/    2519 :                    ;      Then
5353/    2519 :                    ;        Excpt_Stat.SprTbl_Warn:=true
5354/    2519 :                    ;        SetStatus(SprBlk_Warn)
5355/    2519 :                    ;  End
5356/    2519 :                    ;
5357/    2519 :                    ;*****************************************************************
5358/    2519 :
5359/    2519 : 18 E0              SpareCount      ld R1, R0               ; get command
5360/    251B : 56 E1 FC                   and R1, #0FCh           ; check for illegal command
5361/    251E : 6B 05                      jr Z, SprCnt_1
5362/    2520 : 98 E0                      ld R9, R0
5363/    2522 : D6 05 1F                   call Abort
5364/    2525 :                    ;
5365/    2525 : 2C 15              SprCnt_1        ld R2, #SprCount /256
5366/    2527 : 3C 45                      ld R3, #SprCount #256
5367/    2529 : 82 12                      lde R1, @RR2            ; assume spare count increment
5368/    252B : 1E                         inc R1
5369/    252C : A6 E0 00                   cp R0, #0               ; check for Inc_SpareCount
5370/    252F : 6B 12                      jr Z, S_C_Spare
5371/    2531 : A0 E2                      incw RR2                ; get address of BadCount
5372/    2533 : 82 12                      lde R1, @RR2            ; get bad block count
5373/    2535 : A6 E0 01                   cp R0, #1               ; check for Inc_BadBlock
5374/    2538 : 6B 04                      jr Z, S_C_BadInc
5375/    253A : 00 E1                      dec R1                  ; otherwise Decrement bad block count
5376/    253C : 8B 01                      jr S_C_BadEnd
5377/    253E :                    ;
5378/    253E : 1E                 S_C_BadInc      inc R1
5379/    253F : 92 12              S_C_BadEnd      lde @RR2, R1            ; store new count
5380/    2541 : 8B 02                      jr SprCnt_End
5381/    2543 :                    ;
5382/    2543 : 92 12              S_C_Spare       lde @RR2, R1            ; store new count
5383/    2545 : D6 25 4B           SprCnt_End      call Chk_SprCnt
5384/    2548 : 8D 04 F8                   jp Bank_Ret
5385/    254B :                    ;
5386/    254B : 2C 15              Chk_SprCnt      ld R2, #SprCount /256
5387/    254D : 3C 45                      ld R3, #SprCount #256
5388/    254F : 82 02                      lde R0, @RR2            ; get spare count
5389/    2551 : A0 E2                      incw RR2
5390/    2553 : 82 12                      lde R1, @RR2            ; get bd block count
5391/    2555 : 02 01                      add R0, R1
5392/    2557 : A6 E0 47                   cp R0, #76-5            ; check for spare count overflow
5393/    255A : 1B 06                      jr LT, Chk_Spr_Ret
5394/    255C : 46 24 40                   or Excpt_Stat, #SprTbl_Warn
5395/    255F : D6 05 A8                   call SS_SprWarn
5396/    2562 : 8D 04 F8           Chk_Spr_Ret     jp Bank_Ret
5397/    2565 :
5398/    2565 :
5399/    2565 :
5400/    2565 :                    ;*****************************************************************
5401/    2565 :                    ; Module Math.B1.Assem
5402/    2565 :                    ;
5403/    2565 :                    ; This module contains all the special purpose math routines
5404/    2565 :                    ; used in the Widget firmware.
```

```
5405/    2565 :                     ;
5406/    2565 :                     ;       FUNCTION Div3_19(A : 3 BYTES {R12:14}) : 3 BYTES {R0:2}
5407/    2565 :                     ;       FUNCTION Div3_38(A : 3 BYTES {R12:14}) : 3 BYTES {R0:2}
5408/    2565 :                     ;       FUNCTION Div3_76(A : 3 BYTES {R12:14}) : 3 BYTES {R0:2}
5409/    2565 :                     ;       FUNCTION MulR0_m(R0 : BYTE {R0} ) : 3 BYTES {R0:2}
5410/    2565 :                     ;       FUNCTION DivHdsSctrs( A : 3 BYTES {R0:2}) :
5411/    2565 :                     ;                Quotient : BYTE {R0:1}  Remainder : BYTE {R3}
5412/    2565 :                     ;       FUNCTION DivSctrs(A : BYTE {R3}) :
5413/    2565 :                     ;                Quotient : BYTE {R2}    Remainder : BYTE {R3}
5414/    2565 :                     ;
5415/    2565 :                     ;*****************************************************************
5416/    2565 :
5417/    2565 :                     ;*****************************************************************
5418/    2565 :                     ;
5419/    2565 :                     ; Function: Div3_k, Div3_19, Div3_38, Div3_76
5420/    2565 :                     ;
5421/    2565 :                     ;  This function performs the following 24 bit arithmetic operations:
5422/    2565 :                     ;  Div3_19: Result <-- A div 19
5423/    2565 :                     ;  Div3_38: Result <-- A div 38
5424/    2565 :                     ;  Div3_76: Result <-- A div 76
5425/    2565 :                     ;
5426/    2565 :                     ; Inputs:  A: 3 BYTES {R12:14}
5427/    2565 :                     ;
5428/    2565 :                     ; Outputs: Result: 3 BYTES {R0:2}
5429/    2565 :                     ;
5430/    2565 :                     ; Local Variables: Divisor:  3 BYTES {R12:14}
5431/    2565 :                     ;                  Quotient: 3 BYTES {R7:9}
5432/    2565 :                     ;                  Dividend: 3 BYTES {R4:6}
5433/    2565 :                     ;
5434/    2565 :                     ; Algorithm: (div 19)
5435/    2565 :                     ;
5436/    2565 :                     ;  Begin
5437/    2565 :                     ;    Divisor:=$013000 {19*2^(13-1)}
5438/    2565 :                     ;    Quotient:=0
5439/    2565 :                     ;    For i:=1 To 13 Do
5440/    2565 :                     ;     If Dividend>=Divisor
5441/    2565 :                     ;       Then
5442/    2565 :                     ;         Dividend:=Dividend-Divisor
5443/    2565 :                     ;         Quotient:=Quotient*2 +1
5444/    2565 :                     ;       Else Quotient:=Quotient*2
5445/    2565 :                     ;     Divisor:=Divisor div 2
5446/    2565 :                     ;  End
5447/    2565 :                     ;
5448/    2565 :                     ;*****************************************************************
5449/    2565 :
5450/    2565 : E6 4C 04            Div3_76         ld ScrRegC, #004h       ; Divisor:=$04C000
5451/    2568 : E6 4D C0                            ld ScrRegD, #0C0h
5452/    256B : E6 4E 00                            ld ScrRegE, #000h
5453/    256E : 8B 14                               jr Div3_x
5454/    2570 :                     ;
5455/    2570 : E6 4C 02            Div3_38         ld ScrRegC, #002h       ; Divisor:=$026000
5456/    2573 : E6 4D 60                            ld ScrRegD, #060h
5457/    2576 : E6 4E 00                            ld ScrRegE, #000h
5458/    2579 : 8B 09                               jr Div3_x
5459/    257B :                     ;
5460/    257B : E6 4C 01            Div3_19         ld ScrRegC, #001h       ; Divisor:=$013000
5461/    257E : E6 4D 30                            ld ScrRegD, #030h
5462/    2581 : E6 4E 00                            ld ScrRegE, #000h
5463/    2584 :                     ;
5464/    2584 : C9 44               Div3_x          ld ScrReg4, R12         ; pass dividend to routine
5465/    2586 : D9 45                               ld ScrReg5, R13
5466/    2588 : E9 46                               ld ScrReg6, R14
5467/    258A : 70 FD                               push RP                 ; save context
5468/    258C : 31 40                               srp #Wrk_Scr
5469/    258E :                             assume RP:Wrk_Scr
5470/    258E : B0 E7                               clr R7                  ; Quotient:=0
5471/    2590 : B0 E8                               clr R8
5472/    2592 : B0 E9                               clr R9
5473/    2594 : AC 0D                               ld R10, #13             ; For i:=1 To 13 Do
5474/    2596 : CF                  Div3_19_Lp      rcf                     ; clear carry
5475/    2597 : 10 E9                               rlc R9                  ; Quotient:=Quotient*2
5476/    2599 : 10 E8                               rlc R8
5477/    259B : 10 E7                               rlc R7
5478/    259D : 08 E4                               ld R0, R4               ; If Dividend>=Divisor
5479/    259F : 18 E5                               ld R1, R5
5480/    25A1 : 28 E6                               ld R2, R6
5481/    25A3 : D6 03 A5                            call Sub3
5482/    25A6 : 7B 0F                               jr C, Div3_19_Div2
5483/    25A8 :                     ;
5484/    25A8 : 48 E0                               ld R4, R0               ; Dividend:=Dividend-Divisor
5485/    25AA : 58 E1                               ld R5, R1
5486/    25AC : 68 E2                               ld R6, R2
5487/    25AE : 06 E9 01                            add R9, #1              ; Quotient:=Quotient+1
5488/    25B1 : 16 E8 00                            adc R8, #0
5489/    25B4 : 16 E7 00                            adc R7, #0
5490/    25B7 : CF                  Div3_19_Div2    rcf                     ; clear carry flag
5491/    25B8 : C0 EC                               rrc R12                 ; Divisor:=Divisor div 2
5492/    25BA : C0 ED                               rrc R13
5493/    25BC : C0 EE                               rrc R14
5494/    25BE : AA D6                               djnz R10, Div3_19_Lp
5495/    25C0 :                     ;
5496/    25C0 : 50 FD                               pop RP                  ; context switch
5497/    25C2 : 08 47                               ld R0, >ScrReg7         ; pass quotient to caller
5498/    25C4 : 18 48                               ld R1, >ScrReg8
5499/    25C6 : 28 49                               ld R2, >ScrReg9
5500/    25C8 : 8D 04 F8                            jp Bank_Ret
5501/    25CB :
5502/    25CB :
5503/    25CB :                     ;*****************************************************************
5504/    25CB :                     ;
5505/    25CB :                     ; Function: MulR0_m  {multiply R0 by m}
5506/    25CB :                     ;
5507/    25CB :                     ;  This function multiplies the byte in R0 by m where:
5508/    25CB :                     ;  for 10MByte Widget: m=256
5509/    25CB :                     ;  for 20MByte Widget: m=512
5510/    25CB :                     ;  for 30MByte Widget: m=1024
5511/    25CB :                     ;
5512/    25CB :                     ; Inputs:  A: BYTES {R0}
5513/    25CB :                     ;
5514/    25CB :                     ; Outputs: Result: 3 BYTES {R12:14}
5515/    25CB :                     ;
5516/    25CB :                     ;*****************************************************************
5517/    25CB :
5518/    25CB : B0 EE               MulR0_m         clr R14         ; Result:=R0*256
5519/    25CD : D8 E0                               ld R13, R0
5520/    25CF : B0 EC                               clr R12
5521/    25D1 : =>FALSE                     if W_20MB || W_40MB
5522/    25D1 :                                     rlc R12         ; Result:=Result*2
```

```
5523/    25D1 :                                  rlc R13
5524/    25D1 :                                  rlc R14
5525/    25D1 : [5521]                   endif
5526/    25D1 : =>FALSE                  if W_40MB
5527/    25D1 :                                  rlc R12         ; Result:=Result*2
5528/    25D1 :                                  rlc R13
5529/    25D1 :                                  rlc R14
5530/    25D1 : [5526]                   endif
5531/    25D1 : 8D 04 F8                         jp Bank_Ret
5532/    25D4 :
5533/    25D4 :
5534/    25D4 :                  ;*****************************************************************
5535/    25D4 :                  ;
5536/    25D4 :                  ; Function: DivHdsSctrs  {divide by Heads*Sectors}
5537/    25D4 :                  ;
5538/    25D4 :                  ;  This function takes the 24 bit word passed to it and returns
5539/    25D4 :                  ;  a 2 byte integer (representing the cylinder number) and
5540/    25D4 :                  ;  the remainder from the divide.
5541/    25D4 :                  ;
5542/    25D4 :                  ; Inputs:  A: 3 BYTES {R12:14}
5543/    25D4 :                  ;
5544/    25D4 :                  ; Outputs: Cylinder:  WORD {RR0}
5545/    25D4 :                  ;          Remainder: BYTE {R3}
5546/    25D4 :                  ;
5547/    25D4 :                  ;*****************************************************************
5548/    25D4 :
5549/    25D4 :                  DivHdsSctrs
5550/    25D4 : =>TRUE                   if W_10MB
5551/    25D4 : D6 25 70                         call Div3_38            ; return with R0:2 = result
5552/    25D7 : [5550]                   endif
5553/    25D7 : =>FALSE                  if W_20MB || W_40MB
5554/    25D7 :                                  call Div3_76            ;   ScrReg4:6 = remainder
5555/    25D7 : [5553]                   endif
5556/    25D7 : 08 E1                            ld R0, R1               ; HiCyl
5557/    25D9 : 18 E2                            ld R1, R2               ; LoCyl
5558/    25DB : 38 46                            ld R3, >ScrReg6         ; Remainder
5559/    25DD : 8D 04 F8                         jp Bank_Ret
5560/    25E0 :
5561/    25E0 :
5562/    25E0 :                  ;*****************************************************************
5563/    25E0 :                  ;
5564/    25E0 :                  ; Function: DivSctrs  {divide by Sectors}
5565/    25E0 :                  ;
5566/    25E0 :                  ;  This function takes the 1 byte word passed to it and returns
5567/    25E0 :                  ;  a 1 byte integer (representing the head number) and
5568/    25E0 :                  ;  the remainder from the divide (the remainder is the sector #).
5569/    25E0 :                  ;
5570/    25E0 :                  ; Inputs:  A: BYTES {R3}
5571/    25E0 :                  ;
5572/    25E0 :                  ; Outputs: Sector:  BYTE {R2}
5573/    25E0 :                  ;          Remainder: BYTE {R3}
5574/    25E0 :                  ;
5575/    25E0 :                  ; Algorithm:
5576/    25E0 :                  ;
5577/    25E0 :                  ;  Note that there are only 2 heads on the Widget, therefore
5578/    25E0 :                  ;  if A>NumberOfSectors then Head:=1, else Head:=0 and
5579/    25E0 :                  ;  if A>NumberOfSectors then Sector:=A-NumberOfSectors
5580/    25E0 :                  ;   else Sector:=A.
5581/    25E0 :                  ;  Also, the sector is logically remapped to account for
5582/    25E0 :                  ;  different interleaved drives. The mapping table is located
5583/    25E0 :                  ;  in the SpareTable: when the physical sector is founf it's
5584/    25E0 :                  ;  value is used as an index into the Map_Table and the value
5585/    25E0 :                  ;  stored at that location of the table becomes the new sector.
5586/    25E0 :                  ;
5587/    25E0 :                  ;*****************************************************************
5588/    25E0 :
5589/    25E0 : 0C 13            DivSctrs        ld R0, #NbrSctrs        ; load number of sectors into R0
5590/    25E2 : A2 30                            cp R3, R0               ; check for A greater/equal
5591/    25E4 : 1B 06                            jr LT, D_Sctrs1
5592/    25E6 : 2C 01                            ld R2, #1               ; select head 1
5593/    25E8 : 22 30                            sub R3, R0
5594/    25EA : 8B 02                            jr D_Sctrs2
5595/    25EC : B0 E2            D_Sctrs1        clr R2                  ; select head 0
5596/    25EE : AF               D_Sctrs2        ret
5597/    25EF :
5598/    25EF :
5599/    25EF : 70 EE            ReMap_Sector    push R14                ; save regs
5600/    25F1 : 70 EF                            push R15
5601/    25F3 : EC 16                            ld R14, #Map_Table /256
5602/    25F5 : FC 81                            ld R15, #Map_Table #256
5603/    25F7 : 02 F0                            add R15, R0             ; index into table
5604/    25F9 : 16 EE 00                         adc R14, #0
5605/    25FC : 82 0E                            lde R0, @RR14           ; get remapped sector
5606/    25FE : 50 EF                            pop R15
5607/    2600 : 50 EE                            pop R14
5608/    2602 : 8D 04 F8                         jp Bank_Ret
5609/    2605 :
5610/    2605 :
5611/    2605 :
5612/    2605 :                  ;*****************************************************************
5613/    2605 :                  ; Module Srvo1.B1.Assem
5614/    2605 :                  ;
5615/    2605 :                  ; This module contains all the specifications and source code for
5616/    2605 :                  ; controlling the Widget Servo Board (i.e. communication protocol,
5617/    2605 :                  ; seek and head positioning, spare table lookup, etc)
5618/    2605 :                  ;
5619/    2605 :                  ;       PROCEDURE Seek(Cylinder : WORD {RR12}
5620/    2605 :                  ;                Head : BYTE {R14}  Sector : BYTE {R15})
5621/    2605 :                  ;       FUNCTION PosHeads(Wait : BOOLEAN {R8/bit 6}
5622/    2605 :                  ;                Cylinder : WORD {RR12}  Head : BYTE {R14}) : BOOLEAN
5623/    2605 :                  ;       FUNCTION CalcMagDir(Cylinder : WORD {RR12}) :
5624/    2605 :                  ;                BOOLEAN   Magnitude : WORD {R5} Direction : BYTE {R9}
5625/    2605 :                  ;       FUNCTION ServoOk : BOOLEAN  RecalMagDir : BYTE {R0}
5626/    2605 :                  ;       PROCEDURE SelectHead(Head : BYTE {R14}
5627/    2605 :                  ;       PROCEDURE Auto_Offset
5628/    2605 :                  ;       FUNCTION Get_Zone(Cylinder : WORD {RR12})
5629/    2605 :                  ;
5630/    2605 :                  ;*****************************************************************
5631/    2605 :
5632/    2605 :                  ;*****************************************************************
5633/    2605 :                  ;
5634/    2605 :                  ; Procedure: Seek
5635/    2605 :                  ;
5636/    2605 :                  ;  This procedure accepts cylinder, head, and sector values
5637/    2605 :                  ;  and then calls PositionHeads.
5638/    2605 :                  ;
5639/    2605 :                  ; Inputs:  Cylinder: WORD {R12}
5640/    2605 :                  ;          Head:     BYTE {R14}
```

```
5641/    2605 :                     ;          Sector:   BYTE {R15}
5642/    2605 :                     ;
5643/    2605 :                     ; Outputs: none
5644/    2605 :                     ;
5645/    2605 :                     ; Global Variables Changed: Cylinder, Head, Sector
5646/    2605 :                     ;
5647/    2605 :                     ; Algorithm:
5648/    2605 :                     ;
5649/    2605 :                     ;  Begin
5650/    2605 :                     ;    LastSeekAddress:=CurrentSeekAddress
5651/    2605 :                     ;    DiskStat.Parked:=false
5652/    2605 :                     ;    If DiskStat.SeekComplete
5653/    2605 :                     ;      Then
5654/    2605 :                     ;        i:=4
5655/    2605 :                     ;        While i>0 and not(PositionHeads(Wait, Dmt_Seek, Cylinder,
5656/    2605 :                     ;                                        Head, Sector) Do
5657/    2605 :                     ;          If not(Recovery) Then Abort
5658/    2605 :                     ;          i:=i-1
5659/    2605 :                     ;        If i=0 Then Abort
5660/    2605 :                     ;      Else wait up to 1 sec for ServoReady, if timeout Abort
5661/    2605 :                     ;    DiskStat.SeekComplete:=true
5662/    2605 :                     ;    DiskStat.OnTrack:=true
5663/    2605 :                     ;    GlobalCylinder:=Cylinder
5664/    2605 :                     ;    SelectHead( Head )
5665/    2605 :                     ;    GlobalSector:=Sector
5666/    2605 :                     ;    SectorCount:=SectorCount+1
5667/    2605 :                     ;    Chk_Offset
5668/    2605 :                     ;  End
5669/    2605 :                     ;
5670/    2605 :                     ;*********************************************************************
5671/    2605 :
5672/    2605 : D6 22 15            Seek            call Ext_Push
5673/    2608 : E4 52 5C                            ld Lst_HiCyl, Cylinder  ; save last seek address
5674/    260B : E4 53 5D                            ld Lst_LoCyl, Cylinder+1
5675/    260E : E4 54 5E                            ld Lst_Head, Head
5676/    2611 : E4 55 5F                            ld Lst_Sector, Sector
5677/    2614 : 56 56 EF                            and DiskStat, #0FFh-Parked
5678/    2617 : 76 56 02                            tm DiskStat, #SeekComplete      ; check for head/arm motion
5679/    261A : 6B 2E                               jr Z, Seek_Wait
5680/    261C : 1C 04               Seek_ReTry      ld R1, #4               ; four attempts
5681/    261E :                     ;
5682/    261E : 8C 40               Seek_Lp         ld R8, #Wait
5683/    2620 : D6 26 88                            call PosHeads
5684/    2623 : EB 2A                               jr NZ, Seek_End
5685/    2625 : 56 56 FE                            and DiskStat, #0FFh-Offset_On   ; remove any offsets
5686/    2628 : E6 57 90                            ld Seek_Type, #Access_Offset    ; all reseek are with offset
5687/    262B : 76 24 80                            tm Excpt_Stat, #Recovery
5688/    262E : 6B 0D                               jr Z, Seek_Abt
5689/    2630 : 70 E1                               push R1                 ; save counter
5690/    2632 : 2C 00                               ld R2, #20 /256
5691/    2634 : 3C 14                               ld R3, #20 #256
5692/    2636 : D6 01 CB                            call MsWait             ; wait 200 ms before retrying
5693/    2639 : 50 E1                               pop R1                  ; get counter back
5694/    263B : 1A E1                               djnz R1, Seek_Lp
5695/    263D :                     ;
5696/    263D : 0C 00               Seek_Abt        ld R0, #0               ; status byte 0
5697/    263F : 1C 02                               ld R1, #Stat_Srvo
5698/    2641 : D6 04 03                            call SetStatus
5699/    2644 : 46 56 02                            or DiskStat, #SeekComplete
5700/    2647 : D6 05 1F                            call Abort
5701/    264A :                     ;
5702/    264A : D6 26 68            Seek_Wait       call Wait_For_Rdy
5703/    264D : 6B CF                               jr Z, Seek_Lp
5704/    264F :                     ;
5705/    264F : 46 56 82            Seek_End        or DiskStat, #On_Track+SeekComplete
5706/    2652 : C9 52                               ld Cylinder, R12
5707/    2654 : D9 53                               ld Cylinder+1, R13
5708/    2656 : D6 27 17                            call SelectHead
5709/    2659 : F9 55                               ld Sector, R15
5710/    265B : A0 2A                               incw SeekCount
5711/    265D : D6 27 F7                            call Chk_Offset
5712/    2660 : 6B BC                               jr Z, Seek_Lp
5713/    2662 :
5714/    2662 : D6 22 36            Seek_Ret        call Ext_Pop
5715/    2665 : 8D 04 F8                            jp Bank_Ret
5716/    2668 :
5717/    2668 :
5718/    2668 :                     ;*********************************************************************
5719/    2668 :                     ;
5720/    2668 :                     ; Procedure: Wait_For_Rdy
5721/    2668 :                     ;
5722/    2668 :                     ;  This procedure performs a busy wait until SioRdy and ServoRdy.
5723/    2668 :                     ;
5724/    2668 :                     ; Inputs: x: BYTE {R1}
5725/    2668 :                     ;
5726/    2668 :                     ;*********************************************************************
5727/    2668 :
5728/    2668 : 70 E1               Wait_For_Rdy    push R1
5729/    266A : 1C 64                               ld R1, #100             ; wait for up to a second
5730/    266C : 56 FA DF            Seek_Wt         and IRQ, #0FFh-Timer1   ; clear old timer interrupts
5731/    266F : 76 02 40            Seek_Wt_Lp      tm P2, #SioRdy          ; wait for the servo to come back
5732/    2672 : 6B 08                               jr Z, WtRdy_Tmr
5733/    2674 :                     ;
5734/    2674 : D6 03 FC                            call LoadStatus         ; check for ServoReady
5735/    2677 : 76 E0 20                            tm R0, #ServoRdy
5736/    267A : EB 09                               jr NZ, WtRdy_End
5737/    267C :                     ;
5738/    267C : 76 FA 20            WtRdy_Tmr       tm IRQ, #Timer1         ; check for timer interrupt
5739/    267F : 6B EE                               jr Z, Seek_Wt_Lp
5740/    2681 : 00 E1                               dec R1                  ; decrement counter
5741/    2683 : EB E7                               jr NZ, Seek_Wt          ;  and start again
5742/    2685 :                     ;
5743/    2685 : 50 E1               WtRdy_End       pop R1
5744/    2687 : AF                                  ret
5745/    2688 :
5746/    2688 :
5747/    2688 :                     ;*********************************************************************
5748/    2688 :                     ;
5749/    2688 :                     ; Function: PosHeads  {position heads}
5750/    2688 :                     ;
5751/    2688 :                     ;  This function is responsible for positioning the heads of the
5752/    2688 :                     ;  drive; by supplying the cylinder and head value of the desired
5753/    2688 :                     ;  track. PosHeads can be used in one of two modes: 1) as an
5754/    2688 :                     ;  overlapped function where the heads can be in motion while the
5755/    2688 :                     ;  functions returns back to the user or 2) the function will wait
5756/    2688 :                     ;  for the heads to settle before returning to the caller.
5757/    2688 :                     ;
5758/    2688 :                     ; Inputs:  Wait:     BOOLEAN {R8/bit 6}
```

```
5759/    2688 :                     ;          Cylinder: WORD {R12}
5760/    2688 :                     ;          Head:     BYTE {R14}
5761/    2688 :                     ;
5762/    2688 :                     ; Outputs: PositionHeads: BOOLEAN {zero flag is set if servo err}
5763/    2688 :                     ;
5764/    2688 :                     ; Global Variables Changed: Head, Cur_Cyl
5765/    2688 :                     ;
5766/    2688 :                     ; Algorithm:
5767/    2688 :                     ;
5768/    2688 :                     ;  Begin
5769/    2688 :                     ;   DiskStat.Offset_On := False
5770/    2688 :                     ;   SelectHead( 1 )
5771/    2688 :                     ;   IF ServoOk
5772/    2688 :                     ;     THEN
5773/    2688 :                     ;       IF ( CalcMagnitudeDirection( Cylinder, Magnitude, Direction ) <> 0 )
5774/    2688 :                     ;         THEN
5775/    2688 :                     ;           DiskStat.SeekComplete := False
5776/    2688 :                     ;           DiskStat.On_Track := False
5777/    2688 :                     ;           SetDeadManTimer
5778/    2688 :                     ;           ServoStore( IF NOT( Recovery )
5779/    2688 :                     ;                         THEN Access+Direction+
5780/    2688 :                     ;                              Magnitude[ 1 ], Magnitude[ 2 ],
5781/    2688 :                     ;                              0, 0 )
5782/    2688 :                     ;                         ELSE
5783/    2688 :                     ;                           IF ( SeekType = Access_Offset )
5784/    2688 :                     ;                             THEN Access_Offset+Direction+
5785/    2688 :                     ;                                  Magnitude[ 1 ], Magnitude[ 2 ],
5786/    2688 :                     ;                                  AutoOffset, 0 )
5787/    2688 :                     ;                                ELSE
5788/    2688 :                     ;                                  IF NOT( Get_Zone )
5789/    2688 :                     ;                                    THEN Access+Direction+
5790/    2688 :                     ;                                      Magnitude[ 1 ], Magnitude[ 2 ],
5791/    2688 :                     ;                                      0, 0 )
5792/    2688 :                     ;                                    ELSE Access_Offset+Direction+
5793/    2688 :                     ;                                         Magnitude[ 1 ], Magnitude[ 2 ],
5794/    2688 :                     ;                                         Manual_Offset, 0 )
5795/    2688 :                     ;         ClearDeadManTimer
5796/    2688 :                     ;       IF ( SeekType = Access_Offset ) THEN DiskStat.Offset_On := True
5797/    2688 :                     ;      Cur_Cyl := LocalCylinder
5798/    2688 :                     ;    PositionHeads := NOT( LoadStatus.ServoErr )
5799/    2688 :                     ;  End
5800/    2688 :                     ;
5801/    2688 :                     ;*****************************************************************
5802/    2688 :                             assume RP:Wrk_Sys
5803/    2688 : D6 22 15            PosHeads     call Ext_Push           ; save callers variables
5804/    268B : EC 01                          ld R14, #1              ; do all servoing off of head 1
5805/    268D : D6 27 17                       call SelectHead
5806/    2690 : D6 27 5A                       call ServoOk            ; test if Servo is in a reasonable state
5807/    2693 : 6B 6A                          jr Z, PosHds_6          ; leave if servo can't be made useable
5808/    2695 :                     ;
5809/    2695 : D6 27 28                       call CalcMagDir
5810/    2698 : 6B 61                          jr Z, PosHds_5          ; leave if already there
5811/    269A :                     ;
5812/    269A : 56 56 6C                       and DiskStat, #0FFh-SeekComplete-On_Track-Parked-Offset_On
5813/    269D : A6 57 90                       cp Seek_Type, #Access_Offset
5814/    26A0 : 6B 14                          jr Z, GtSk_LdAO
5815/    26A2 : 76 24 80                       tm Excpt_Stat, #Recovery
5816/    26A5 : E6 42 00                       ld ScrReg2, #0          ; default is no offsets
5817/    26A8 : 6B 14                          jr Z, GtSk_LdNo
5818/    26AA :                     ;
5819/    26AA : D6 28 07                       call Get_Zone
5820/    26AD : E6 42 00                       ld ScrReg2, #0          ; default is no offsets
5821/    26B0 : 1B 0C                          jr LT, GtSk_LdNo
5822/    26B2 :                     ;
5823/    26B2 : 09 42                          ld ScrReg2, R0          ; set amount of offset
5824/    26B4 : 8B 03                          jr GS_LdAO1
5825/    26B6 :                     ;
5826/    26B6 : E6 42 40            GtSk_LdAO    ld ScrReg2, #Off_Auto
5827/    26B9 : E6 40 90            GS_LdAO1     ld ScrReg0, #Access_Offset
5828/    26BC : 8B 03                          jr GS_LdNo1
5829/    26BE :                     ;
5830/    26BE : E6 40 80            GtSk_LdNo    ld ScrReg0, #Access
5831/    26C1 : 31 40               GS_LdNo1     srp #Wrk_Scr
5832/    26C3 :                             assume RP:Wrk_Scr
5833/    26C3 : 44 19 E0                       or R0, Wrk_Sys+9        ; plus direction
5834/    26C6 : 44 15 E0                       or R0, Wrk_Sys+5        ; plus MSB magnitude
5835/    26C9 : 18 16                          ld R1, Wrk_Sys+6        ; LSB magnitude
5836/    26CB : 3C 80                          ld R3, #S_Rate_57_6
5837/    26CD : 31 10                          srp #Wrk_Sys
5838/    26CF :                             assume RP:Wrk_Sys
5839/    26CF : D6 04 13                       call Set_Dmt
5840/    26D2 : D6 28 37                       call ServoCmnd
5841/    26D5 : 8F                             di                      ; clear Dead_Man_Timer
5842/    26D6 : A6 57 90                       cp Seek_Type, #Access_Offset    ; check for auto_offset
5843/    26D9 : EB 03                          jr NZ, PosHds_4
5844/    26DB :                     ;
5845/    26DB : 46 56 01                       or DiskStat, #Offset_On
5846/    26DE : EC 61               PosHds_4     ld R14, #25000 /256     ; intermediate timer of 1sec
5847/    26E0 : FC A8                          ld R15, #25000 #256
5848/    26E2 : D6 03 FC            PosHds_42    call LoadStatus         ; sample ServoError
5849/    26E5 : 76 E0 10                       tm R0, #ServoErr
5850/    26E8 : EB 11                          jr NZ, PosHds_5         ; if servo error then exit
5851/    26EA :
5852/    26EA : 76 E8 40            PosHds_3     tm R8, #Wait            ; IF Wait AND NOT( ServoRdy )
5853/    26ED : 6B 0C                          jr Z, PosHds_5
5854/    26EF : 80 EE                          decw RR14
5855/    26F1 : 6B 08                          jr Z, PosHds_5
5856/    26F3 : 76 E0 20                       tm R0, #ServoRdy        ; THEN loop until timeout OR ServoRdy
5857/    26F6 : 6B EA                          jr Z, PosHds_42
5858/    26F8 :                     ;
5859/    26F8 : 46 56 02                       or DiskStat, #SeekComplete
5860/    26FB : C9 50               PosHds_5     ld Cur_Cyl, R12
5861/    26FD : D9 51                          ld Cur_Cyl+1, R13
5862/    26FF :                     ;
5863/    26FF : D6 22 36            PosHds_6     call Ext_Pop            ; get caller's variables back
5864/    2702 : D6 27 08                       call Chk_SStat
5865/    2705 : 8D 04 F8                       jp Bank_Ret
5866/    2708 :                     ;
5867/    2708 : D6 03 FC            Chk_SStat    call LoadStatus         ; get current servo status
5868/    270B : 18 E0                          ld R1, R0               ; get copies of both ServoErr and ServoRdy
5869/    270D : 60 E0                          com R0                  ; ServoErr := NOT( ServoErr )
5870/    270F : 90 E0                          rl R0                   ; get ServoErr in same position as ServoRdy
5871/    2711 : 52 01                          and R0, R1              ; return AND( ServoRdy, NOT( ServoErr ) )
5872/    2713 : 56 E0 20                       and R0, #ServoRdy       ; set zero flag
5873/    2716 : AF                             ret
5874/    2717 :
5875/    2717 :
5876/    2717 :                     ;*****************************************************************
```

```
5877/    2717 :                     ;
5878/    2717 :                     ; Procedure: SelectHead
5879/    2717 :                     ;
5880/    2717 :                     ;  This procedure is responsible for selecting the correct head
5881/    2717 :                     ;  on the disk.
5882/    2717 :                     ;
5883/    2717 :                     ; Inputs: Head: BYTE {R14}
5884/    2717 :                     ;
5885/    2717 :                     ; Outputs: none
5886/    2717 :                     ;
5887/    2717 :                     ;*********************************************************************
5888/    2717 :
5889/    2717 : 42 EE               SelectHead      or R14, R14             ; test for Head0 or Head1
5890/    2719 : 6B 05                               jr Z, Sel_Head0
5891/    271B : 46 03 20                            or P3, #Hs0             ; select head 1
5892/    271E : 8B 03                               jr SelHd_End
5893/    2720 : 56 03 DF            Sel_Head0       and P3, #0FFh-Hs0       ; select head 0
5894/    2723 : E9 54               SelHd_End       ld Head, R14
5895/    2725 : 8D 04 F8            SelHd_Ret       jp Bank_Ret
5896/    2728 :
5897/    2728 :
5898/    2728 :                     ;*********************************************************************
5899/    2728 :                     ;
5900/    2728 :                     ; Function: CalcMagDir  {Calculate Magnitude and Direction}
5901/    2728 :                     ;
5902/    2728 :                     ;  This function takes the current cylinder number (from the
5903/    2728 :                     ;  conversion of the logical block) and generates a magnitude and
5904/    2728 :                     ;  direction from the the position that the heads are currently at
5905/    2728 :                     ;  (the servo needs to know a RELATIVE distance, not an absolute
5906/    2728 :                     ;  distance).
5907/    2728 :                     ;
5908/    2728 :                     ; Inputs:       LocalCylinder : WORD {R12:13}
5909/    2728 :                     ;
5910/    2728 :                     ; Outputs:      CalcMagDir : BOOLEAN {R4}
5911/    2728 :                     ;               Magnitude : WORD {R5:6}
5912/    2728 :                     ;               Direction : BYTE {R9}
5913/    2728 :                     ;
5914/    2728 :                     ; Local Variables:      Temp : BOOLEAN {R4}
5915/    2728 :                     ;                       GlobalPTR : PTR {R2:3}
5916/    2728 :                     ;
5917/    2728 :                     ; Algorithm:
5918/    2728 :                     ;
5919/    2728 :                     ; BEGIN
5920/    2728 :                     ;   IF LocalCylinder <> GlobalCylinder
5921/    2728 :                     ;     THEN
5922/    2728 :                     ;       IF LocalCylinder > GlobalCylinder
5923/    2728 :                     ;         THEN
5924/    2728 :                     ;           Direction := Positive
5925/    2728 :                     ;           Magnitude := LocalCylinder - GlobalCylinder
5926/    2728 :                     ;         ELSE
5927/    2728 :                     ;           Direction := Negative
5928/    2728 :                     ;           Magnitude := GlobalCylinder - LocalCylinder
5929/    2728 :                     ;       Temp := True
5930/    2728 :                     ;     ELSE
5931/    2728 :                     ;       Temp := False
5932/    2728 :                     ;   CalcMagDir := Temp
5933/    2728 :                     ; END
5934/    2728 :                     ;
5935/    2728 :                     ;*********************************************************************
5936/    2728 :
5937/    2728 : 18 50               CalcMagDir      ld R1, Cur_Cyl          ; get current cylinder
5938/    272A : 28 51                               ld R2, Cur_Cyl+1
5939/    272C : 22 2D                               sub R2, R13             ; subtract target cylinder
5940/    272E : 32 1C                               sbc R1, R12
5941/    2730 : B0 E4                               clr R4                  ; assume no seek needed
5942/    2732 : 1B 10                               jr LT, C_MagDir_Else    ; negative --> go backwards
5943/    2734 :                     ;
5944/    2734 : B0 E5                               clr R5                  ; assume 0 track seek
5945/    2736 : B0 E6                               clr R6
5946/    2738 : 9C 00                               ld R9, #0               ; set direction negative
5947/    273A : B0 E3                               clr R3                  ; check for no seek condition
5948/    273C : 42 32                               or R3, R2
5949/    273E : 42 31                               or R3, R1
5950/    2740 : 6B 15                               jr Z, C_MagDir_Ret      ; distance is zero!
5951/    2742 : 8B 0C                               jr S_Glbl_Cyl
5952/    2744 :                     ;
5953/    2744 : 60 E2               C_MagDir_Else   com R2                  ; 2's complement sutraction result
5954/    2746 : 60 E1                               com R1
5955/    2748 : 06 E2 01                            add R2, #1
5956/    274B : 16 E1 00                            adc R1, #0
5957/    274E : 9C 04                               ld R9, #4               ; set direction positive
5958/    2750 :                     ;
5959/    2750 : 58 E1               S_Glbl_Cyl      ld R5, R1               ; load magnitude
5960/    2752 : 68 E2                               ld R6, R2
5961/    2754 : 46 E4 01                            or R4, #1               ; set Seek
5962/    2757 :                     ;
5963/    2757 : 8D 04 F8            C_MagDir_Ret    jp Bank_Ret
5964/    275A :
5965/    275A :
5966/    275A :                     ;*********************************************************************
5967/    275A :                     ;
5968/    275A :                     ; Function: ServoOk
5969/    275A :                     ;
5970/    275A :                     ;  This function is responsible for determining if the servo is in a
5971/    275A :                     ;  reasonable state to perform commands. In other words if ServoReady
5972/    275A :                     ;  and NOT(ServoError) then the servo is healthy, wealthy and wise.
5973/    275A :                     ;  If, on the other hand, ServoError is active then the state of
5974/    275A :                     ;  ServoReady determines the type of action to perform to try to get
5975/    275A :                     ;  the servo back to a nice state:
5976/    275A :                     ;
5977/    275A :                     ;  ServoError and ServoReady: Read Status
5978/    275A :                     ;  ServorError and Not(ServoReady): do Data Recal
5979/    275A :                     ;
5980/    275A :                     ; Inputs: none
5981/    275A :                     ;
5982/    275A :                     ; Outputs:      ServOk: BOOLEAN {Zero Flag, True if NOT(ServoOk) }
5983/    275A :                     ;               RecalcMagDir: BYTE {R0}
5984/    275A :                     ;
5985/    275A :                     ; Local Variables:      Retry   : BYTE {R15}
5986/    275A :                     ;                       SioRetry: BYTE {R8}
5987/    275A :                     ;                       Done    : BOOLEAN { }
5988/    275A :                     ;                       ControllerStatusPTR : PTR {RR12}
5989/    275A :                     ;
5990/    275A :                     ; Algorithm:
5991/    275A :                     ;
5992/    275A :                     ; BEGIN
5993/    275A :                     ;   IF Recovery
5994/    275A :                     ;     THEN
```

```
5995/    275A :                     ;       IF ServoError
5996/    275A :                     ;         THEN
5997/    275A :                     ;           IF Write_Operation THEN WrBuf_To_Buf2
5998/    275A :                     ;           Restore( DataRecal )
5999/    275A :                     ;           Buf2_To_WrBuf
6000/    275A :                     ;         ELSE
6001/    275A :                     ;           IF NOT( ServoReady )
6002/    275A :                     ;             THEN
6003/    275A :                     ;               IF Write_Operation THEN WrBuf_To_Buf2
6004/    275A :                     ;               S_Reset
6005/    275A :                     ;               Buf2_To_WrBuf
6006/    275A :                     ;   ServoOk := NOT( ServoError )
6007/    275A :                     ; END
6008/    275A :                     ;
6009/    275A :                     ;*********************************************************************
6010/    275A :
6011/    275A : 76 24 80            ServoOk         tm Excpt_Stat, #Recovery        ; IF Recovery
6012/    275D : 6B 2A                               jr Z, S_Ok_End
6013/    275F :                     ;
6014/    275F : D6 03 FC                            call LoadStatus
6015/    2762 : 76 E0 10                            tm R0, #ServoErr        ; IF ServoError
6016/    2765 : 6B 0D                               jr Z, SOk_ChkRdy
6017/    2767 :                     ;
6018/    2767 : D6 27 8F                            call Chk_MvWrData       ; save write data if Write_Op
6019/    276A : 0C 40                               ld R0, #DataRecal       ; IF ServoErr AND NOT( ServoRdy )
6020/    276C : D6 28 E2                            call Restore            ;  THEN Restore
6021/    276F : D6 20 70                            call Buf2_To_WrBuf      ; assume write_op
6022/    2772 : 8B 15                               jr S_Ok_End
6023/    2774 :                     ;
6024/    2774 : 76 E0 20            SOk_ChkRdy      tm R0, #ServoRdy        ; IF NOT( ServoErr )
6025/    2777 : EB 10                               jr NZ, S_Ok_End         ; AND ServoRdy
6026/    2779 :                     ;
6027/    2779 : D6 27 8F            SOk_Park        call Chk_MvWrData       ; save write data if Write_Op
6028/    277C : D6 29 A2                            call ResetServo         ; otherwise un-park servo
6029/    277F : 0C 40                               ld R0, #DataRecal
6030/    2781 : D6 28 E2                            call Restore
6031/    2784 : D6 20 70                            call Buf2_To_WrBuf      ; assume write_op
6032/    2787 : 8B 00                               jr S_Ok_End
6033/    2789 :                     ;
6034/    2789 : D6 27 08            S_Ok_End        call Chk_SStat
6035/    278C : 8D 04 F8                            jp Bank_Ret
6036/    278F :                     ;
6037/    278F :                     ;*******************
6038/    278F :                     ;
6039/    278F : 0C 00               Chk_MvWrData    ld R0, #0               ; status byte 0
6040/    2791 : 1C 02                               ld R1, #Stat_Srvo
6041/    2793 : D6 04 03                            call SetStatus
6042/    2796 : 76 56 20                            tm DiskStat, #Wr_Op     ; check for write_op
6043/    2799 : 6B 03                               jr Z, MvWr_End
6044/    279B : D6 20 65                            call WrBuf_To_Buf2
6045/    279E : AF                  MvWr_End        ret
6046/    279F :
6047/    279F :
6048/    279F :                     ;*********************************************************************
6049/    279F :                     ;
6050/    279F :                     ; Procedure: Auto_Offset
6051/    279F :                     ;
6052/    279F :                     ;  This procedure is used to set auto_offset to the servo processor.
6053/    279F :                     ;
6054/    279F :                     ; Inputs: none
6055/    279F :                     ;
6056/    279F :                     ; Outputs: none
6057/    279F :                     ;
6058/    279F :                     ; Algorithm:
6059/    279F :                     ;
6060/    279F :                     ; BEGIN
6061/    279F :                     ;   Temp := Head
6062/    279F :                     ;   SelectHead( 1 )
6063/    279F :                     ;   ServoCmnd( Offset, 0, Off_Auto, S_Rate_57_6 )
6064/    279F :                     ;   @Get_Zone := ServoStatus( 1 ).Offset_Value
6065/    279F :                     ;   DiskStat.Offset_On := True
6066/    279F :                     ; END
6067/    279F :                     ;
6068/    279F :                     ;*********************************************************************
6069/    279F :
6070/    279F : D6 22 15            Auto_Offset     call Ext_Push
6071/    27A2 : 70 54                               push Head               ; save the current head value
6072/    27A4 : EC 01                               ld R14, #1              ; select head 1 for offsetting
6073/    27A6 : D6 27 17                            call SelectHead
6074/    27A9 : D6 04 13                            call Set_Dmt            ; set deadman timer
6075/    27AC : 31 40                               srp #Wrk_Scr
6076/    27AE :                                 assume RP:Wrk_Scr
6077/    27AE : 0C 90                               ld R0, #Access_Offset
6078/    27B0 : 1C 00                               ld R1, #0
6079/    27B2 : 2C 40                               ld R2, #Off_Auto
6080/    27B4 : 3C 80                               ld R3, #S_Rate_57_6
6081/    27B6 : 31 10                               srp #Wrk_Sys
6082/    27B8 :                                 assume RP:Wrk_Sys
6083/    27B8 : D6 28 37                            call ServoCmnd
6084/    27BB :                     ; 1A44          call Wait_For_Rdy
6085/    27BB : D6 03 FC                            call LoadStatus
6086/    27BE : 31 40                               srp #Wrk_Scr
6087/    27C0 :                                 assume RP:Wrk_Scr
6088/    27C0 : 0C 00                               ld R0, #ReadStatus
6089/    27C2 : 1C 00                               ld R1, #0
6090/    27C4 : 2C 00                               ld R2, #0
6091/    27C6 : 3C 01                               ld R3, #1               ; read servo stat 1 to find offset value
6092/    27C8 : 31 10                               srp #Wrk_Sys
6093/    27CA :                                 assume RP:Wrk_Sys
6094/    27CA : D6 28 5D                            call ServoStatus
6095/    27CD : D6 04 1F                            call Clr_Dmt
6096/    27D0 : 2C 14                               ld R2, #(SStatus0+1) /256       ; get offset amount
6097/    27D2 : 3C B7                               ld R3, #(SStatus0+1) #256
6098/    27D4 : 82 02                               lde R0, @RR2
6099/    27D6 : B6 E0 1F                            xor R0, #1Fh            ; get correct polarity
6100/    27D9 : 56 E0 3F                            and R0, #3Fh            ; mask off unwanted stuff
6101/    27DC : 70 E0                               push R0
6102/    27DE :                     ;
6103/    27DE : C8 52                               ld R12, Cylinder
6104/    27E0 : D8 53                               ld R13, Cylinder+1
6105/    27E2 : D6 28 07                            call Get_Zone
6106/    27E5 : 50 E0                               pop R0
6107/    27E7 : 92 02                               lde @RR2, R0            ; store new value
6108/    27E9 :                     ;
6109/    27E9 : 50 EE               Auto_Off_Ret    pop R14                 ; re-select original head
6110/    27EB : D6 27 17                            call SelectHead
6111/    27EE : D6 22 36                            call Ext_Pop
6112/    27F1 : 46 56 01                            or DiskStat, #Offset_On
```

```
6113/    27F4 : 8D 04 F8                        jp Bank_Ret
6114/    27F7 :                          ;
6115/    27F7 :                          ;*****************
6116/    27F7 :                          ;
6117/    27F7 : A6 57 90                 Chk_Offset      cp Seek_Type,#Access_Offset     ; check if auto offset needed
6118/    27FA : EB 08                                    jr NZ, Chk_Off_Ret
6119/    27FC : 76 56 01                 ChkOff_NoOff    tm DiskStat, #Offset_On         ; THEN check for offset already on
6120/    27FF : EB 03                                    jr NZ, Chk_Off_Ret
6121/    2801 : D6 27 9F                                 call Auto_Offset                ; otherwise Auto_Offset
6122/    2804 : 8D 04 F8                 Chk_Off_Ret     jp Bank_Ret
6123/    2807 :
6124/    2807 :
6125/    2807 :                          ;*********************************************************************
6126/    2807 :                          ;
6127/    2807 :                          ; Function: Get_Zone
6128/    2807 :                          ;
6129/    2807 :                          ;  This function is responsible for identifying the zone that the
6130/    2807 :                          ;  heads are currently positioned in; a zone being a section of the
6131/    2807 :                          ;  drive that is assumed to have a homogeneous servo offset
6132/    2807 :                          ;  characteristic.
6133/    2807 :                          ;
6134/    2807 :                          ; Inputs: Cylinder: WORD {RR12}
6135/    2807 :                          ;
6136/    2807 :                          ; Outputs:      Get_Zone : BYTE {R0},
6137/    2807 :                          ;               Get_Zone1 : BOOLEAN {zero flag is set if servo err}
6138/    2807 :                          ;
6139/    2807 :                          ; Algorithm:
6140/    2807 :                          ;
6141/    2807 :                          ; BEGIN
6142/    2807 :                          ;   Get_Zone := Zone_Table[ Cylinder/NbrZones ]
6143/    2807 :                          ;   IF ( Get_Zone <= MinOffset )
6144/    2807 :                          ;     THEN Get_Zone1 := False
6145/    2807 :                          ;     ELSE GEt_Zone1 := True
6146/    2807 :                          ; END
6147/    2807 :                          ;
6148/    2807 :                          ;*********************************************************************
6149/    2807 :
6150/    2807 : 08 EC                    Get_Zone        ld R0, R12              ; use seek address for zone calc
6151/    2809 : 18 ED                                    ld R1, R13
6152/    280B : 2C 05                                    ld R2, #ZoneShift
6153/    280D : CF                                       rcf
6154/    280E : C0 E0                    GtZn_Lp         rrc R0                  ; rotate down the cylinder value
6155/    2810 : C0 E1                                    rrc R1
6156/    2812 : 2A FA                                    djnz R2, GtZn_Lp
6157/    2814 : 2C 16                                    ld R2, #Zone_Table /256
6158/    2816 : 3C 9A                                    ld R3, #Zone_Table #256
6159/    2818 : 02 31                                    add R3, R1              ; add offset to table
6160/    281A : 16 E2 00                                 adc R2, #0
6161/    281D : 82 02                                    lde R0, @RR2            ; get offset value
6162/    281F : 76 E0 20                                 tm R0, #Dir_Frwd        ; mask all but sign bit
6163/    2822 :                          ; 1A44          ld R1, #Off_Dir_Frwd    ; assume forward
6164/    2822 : 1C 00                                    ld R1, #Off_Dir_Rev
6165/    2824 : EB 02                                    jr NZ, GtZn_Push
6166/    2826 :                          ; 1A44          ld R1, #Off_Dir_Rev     ; otherwise reverse
6167/    2826 : 1C 80                                    ld R1, #Off_Dir_Frwd
6168/    2828 : 56 E0 1F                 GtZn_Push       and R0, #1Fh
6169/    282B : A6 E0 0A                                 cp R0, #MinOffset       ; check for lower bound on offset
6170/    282E : 70 FC                                    push FLAGS
6171/    2830 : 42 01                                    or R0, R1               ; merge sign and magnitude
6172/    2832 : 50 FC                                    pop FLAGS
6173/    2834 : 8D 04 F8                                 jp Bank_Ret
6174/    2837 :
6175/    2837 :
6176/    2837 :
6177/    2837 :                          ;*********************************************************************
6178/    2837 :                          ; Module Srvo2.B1.Assem
6179/    2837 :                          ;
6180/    2837 :                          ; This module holds those routines that make up most of the
6181/    2837 :                          ; auxiliary servo commands.
6182/    2837 :                          ;
6183/    2837 :                          ;       PROCEDURE ServoCmnd(CommandString : 4 BYTES {R0:3})
6184/    2837 :                          ;       PROCEDURE ServoStatus(CommandString : 4 BYTES {R0:3}
6185/    2837 :                          ;                             BufferPtr : PTR {RR14})
6186/    2837 :                          ;       FUNCTION ServoStore : BOOLEAN
6187/    2837 :                          ;       FUNCTION ServoLoad(BufferPtr : PTR {RR2}) : BOOLEAN
6188/    2837 :                          ;       FUNCTION Restore(RecalType : BYTE {R0}) : BOOLEAN
6189/    2837 :                          ;       PROCEDURE SrvoRcvry
6190/    2837 :                          ;       PROCEDURE Load_SrvoCmnd
6191/    2837 :                          ;       PROCEDURE Park_Heads
6192/    2837 :                          ;       PROCEDURE ResetServo
6193/    2837 :                          ;
6194/    2837 :                          ;*********************************************************************
6195/    2837 :
6196/    2837 :                          ;*********************************************************************
6197/    2837 :                          ;
6198/    2837 :                          ; Procedure: SrvoCmnd  {send the servo a command}
6199/    2837 :                          ;
6200/    2837 :                          ; Inputs:  CommandString: 4 BYTES {R0:3}
6201/    2837 :                          ;
6202/    2837 :                          ; Outputs: none
6203/    2837 :                          ;
6204/    2837 :                          ; Global Variables Changed: SrvoCmndBuffer
6205/    2837 :                          ;
6206/    2837 :                          ; Local Variables: SioRetry: BYTE {R8}
6207/    2837 :                          ;
6208/    2837 :                          ; Algorithm:
6209/    2837 :                          ;
6210/    2837 :                          ;  Begin
6211/    2837 :                          ;    SrvoCmndBuffer[CommandByte]:=R0
6212/    2837 :                          ;    SrvoCmndBuffer[LoDiffByte]:=R1
6213/    2837 :                          ;    SrvoCmndBuffer[OffsetByte]:=R2
6214/    2837 :                          ;    SrvoCmndBuffer[StatusByte]:=R3
6215/    2837 :                          ;    SioRetry:=4
6216/    2837 :                          ;    While not(ServoStore(CommandString) And SioRetry>0 Do
6217/    2837 :                          ;      SioRetry:=SioRetry-1
6218/    2837 :                          ;    If SioRetry=0 Abort
6219/    2837 :                          ;  End
6220/    2837 :                          ;
6221/    2837 :                          ;*********************************************************************
6222/    2837 :
6223/    2837 : 31 30                    ServoCmnd       srp #Wrk_Sys2
6224/    2839 :                                  assume RP:Wrk_Sys2
6225/    2839 : D6 29 7B                                 call Load_SrvoCmnd
6226/    283C : 8C 04                    Srv_C_Lp1       ld R8, #4               ; SioRetry:=4
6227/    283E : 48 40                                    ld R4, ScrReg0          ; save command
6228/    2840 : 58 41                                    ld R5, ScrReg1
6229/    2842 : A8 42                                    ld R10, ScrReg2
6230/    2844 : B8 43                                    ld R11, ScrReg3
```

```
6231/    2846 :                         ;
6232/    2846 : D6 28 7F                Srv_C_Lp        call ServoStore
6233/    2849 : EB 0D                                   jr NZ, Srv_C_End
6234/    284B : 49 40                                   ld ScrReg0, R4          ; restore command
6235/    284D : 59 41                                   ld ScrReg1, R5
6236/    284F : A9 42                                   ld ScrReg2, R10
6237/    2851 : B9 43                                   ld ScrReg3, R11
6238/    2853 : 8A F1                                   djnz R8, Srv_C_Lp
6239/    2855 : D6 05 1F                                call Abort
6240/    2858 : 31 10                   Srv_C_End       srp #Wrk_Sys
6241/    285A :                                 assume RP:Wrk_Sys
6242/    285A : 8D 04 F8                                jp Bank_Ret
6243/    285D :
6244/    285D :
6245/    285D :                         ;*****************************************************************
6246/    285D :                         ;
6247/    285D :                         ; Procedure: ServoStatus  {read a status location from the servo}
6248/    285D :                         ;
6249/    285D :                         ; Inputs:  CommandString: 4 BYTES {R0:3}
6250/    285D :                         ;          BufferPtr:     PTR {RR14}
6251/    285D :                         ;
6252/    285D :                         ; Outputs: none
6253/    285D :                         ;
6254/    285D :                         ; Global Variables Changed: SrvoCmndBuffer
6255/    285D :                         ;                           @BufferPtr
6256/    285D :                         ;
6257/    285D :                         ; Local Variables: SioRetry: BYTE {R8}
6258/    285D :                         ;
6259/    285D :                         ; Algorithm:
6260/    285D :                         ;
6261/    285D :                         ;  Begin
6262/    285D :                         ;    ServoCmnd
6263/    285D :                         ;    If not(ServoLoad(BufferPtr)) Then Abort
6264/    285D :                         ;  End
6265/    285D :                         ;
6266/    285D :                         ;*****************************************************************
6267/    285D :
6268/    285D : 70 E8                   ServoStatus     push R8                 ; save for caller
6269/    285F : 8C 04                                   ld R8, #4               ; retry 4 times
6270/    2861 : D6 28 37                Servo_StLp1     call ServoCmnd
6271/    2864 : D6 28 BC                                call ServoLoad
6272/    2867 : 6B 11                                   jr Z, Srv_St_End
6273/    2869 : E4 34 40                                ld ScrReg0, Wrk_Sys2+4
6274/    286C : E4 35 41                                ld ScrReg1, Wrk_Sys2+5
6275/    286F : E4 3A 42                                ld ScrReg2, Wrk_Sys2+10
6276/    2872 : E4 3B 43                                ld ScrReg3, Wrk_Sys2+11
6277/    2875 : 8A EA                                   djnz R8, Servo_StLp1
6278/    2877 : D6 05 1F                                call Abort
6279/    287A : 50 E8                   Srv_St_End      pop R8
6280/    287C : 8D 04 F8                                jp Bank_Ret
6281/    287F :
6282/    287F :
6283/    287F :                         ;*****************************************************************
6284/    287F :                         ;
6285/    287F :                         ; Function: ServoStore
6286/    287F :                         ;
6287/    287F :                         ;  The function of this routine is to transmit a command string
6288/    287F :                         ;  to the Servo Processor. The command string is expected to
6289/    287F :                         ;  reside in the global variable SrvoCmndBuffer (thus providing a
6290/    287F :                         ;  record of the last command sent to the servo). ServoStore then
6291/    287F :                         ;  attempts to complete the transmission to the Servo, and returns
6292/    287F :                         ;  a boolean variable indicating whether the transmission was
6293/    287F :                         ;  completed or not.
6294/    287F :                         ;
6295/    287F :                         ; Inputs:  Parent: BYTE {R1}
6296/    287F :                         ;
6297/    287F :                         ; Outputs: ServoStore: BOOLEAN {zero flag, true if transmission failed}
6298/    287F :                         ;
6299/    287F :                         ; Local Variables: Retry: BYTE {R6}
6300/    287F :                         ;                  i:     BYTE {R4}
6301/    287F :                         ;                  j:     BYTE {R5}
6302/    287F :                         ;
6303/    287F :                         ; Algorithm:
6304/    287F :                         ;
6305/    287F :                         ;  Begin
6306/    287F :                         ;    SrvoCmndBuffer[CheckByte]:=GenerateCheckByte(PTR(SrvoCmndBuffer),
6307/    287F :                         ;                                                 Length(SrvoCmndBuffer))
6308/    287F :                         ;    Retry:=10
6309/    287F :                         ;    Repeat
6310/    287F :                         ;      For i:=1 To 5 Do
6311/    287F :                         ;        While IRQ.Serial_Output=false Or not(SioReady) Do
6312/    287F :                         ;          Begin End
6313/    287F :                         ;        Sio.Data:=SrvoCmndBuffer[i]
6314/    287F :                         ;      Retry:=Retry-1
6315/    287F :                         ;      For j:=1 To (value for 250µs wait) Do Begin End
6316/    287F :                         ;    Until not(SioReady) Or Retry=0
6317/    287F :                         ;    For j:=1 To (value for 400µs wait) Do Begin End
6318/    287F :                         ;      If Retry=0
6319/    287F :                         ;        Then ServoStore:=false
6320/    287F :                         ;        Else ServoStore:=true
6321/    287F :                         ;  End
6322/    287F :                         ;
6323/    287F :                         ;*****************************************************************
6324/    287F :
6325/    287F : 70 FD                   ServoStore      push RP                 ; save context
6326/    2881 : 31 40                                   srp #Wrk_Scr
6327/    2883 :                                 assume RP:Wrk_Scr
6328/    2883 : EC 14                                   ld R14, #SrvoCmndBuf /256
6329/    2885 : FC B1                                   ld R15, #SrvoCmndBuf #256
6330/    2887 : 8C 04                                   ld R8, #S_Cmnd_Len-1
6331/    2889 : D6 06 2E                                call Gen_Chk_Byte
6332/    288C : 6C 0A                                   ld R6, #10              ; Retry:=10
6333/    288E : EC 14                   Srvo_St_Rpt     ld R14, #SrvoCmndBuf /256
6334/    2890 : FC B1                                   ld R15, #SrvoCmndBuf #256
6335/    2892 : 4C 05                                   ld R4, #S_Cmnd_Len
6336/    2894 :                         ;
6337/    2894 : 76 02 40                Srvo_St_1       tm P2, #SioRdy          ; While not(SioReady)
6338/    2897 : 6B FB                                   jr Z, Srvo_St_1
6339/    2899 :                         ;
6340/    2899 : 56 FA E7                                and IRQ, #0FFh-Serial_Out-Serial_In     ; clear old interrupts
6341/    289C : 82 0E                                   lde R0, @RR14           ; get a command byte
6342/    289E : 09 F0                                   ld sio, R0              ; send it to servo
6343/    28A0 : A0 EE                                   incw RR14               ; point to next command byte
6344/    28A2 :                         ;
6345/    28A2 : 76 FA 10                Srvo_St_2       tm IRQ, #Serial_Out     ; While IRQ.Serial_Out=false
6346/    28A5 : 6B FB                                   jr Z, Srvo_St_2
6347/    28A7 :                         ;
6348/    28A7 : 4A EB                                   djnz R4, Srvo_St_1      ; loop till all command bytes are sent
```

```
6349/    28A9 : 56 FA EF                         and IRQ, #0FFh-Serial_Out       ; clear old interrupts
6350/    28AC : 5C 50                            ld R5, #80
6351/    28AE : 5A FE                Srvo_Wt_1   djnz R5, Srvo_Wt_1      ; do a 250µs wait
6352/    28B0 : 76 02 40                         tm P2, #SioRdy          ; if not(SioReady) then Servo
6353/    28B3 : 6B 02                            jr Z, Srvo_St_Exit      ;  took the bytes and is munchin' on 'em
6354/    28B5 : 6A D7                            djnz R6, Srvo_St_Rpt    ; Retry:=Retry-1
6355/    28B7 :                      ;
6356/    28B7 : 42 66                Srvo_St_Exit    or R6, R6               ; test for Retry=0
6357/    28B9 : 50 FD                            pop RP                  ; return to original context
6358/    28BB : AF                               ret
6359/    28BC :
6360/    28BC :
6361/    28BC :                      ;*****************************************************************
6362/    28BC :                      ;
6363/    28BC :                      ; Function: ServoLoad
6364/    28BC :                      ;
6365/    28BC :                      ;  This function is responsible for reading a status back
6366/    28BC :                      ;  from the servo. The bytes that it receives are stored into a
6367/    28BC :                      ;  buffer that is passed into the routine in the form of a PTR.
6368/    28BC :                      ;  ServoLoad passes back to the caller a boolean variable describing
6369/    28BC :                      ;  whether the checkbyte was valid (i.e. the transmission was a
6370/    28BC :                      ;  success).
6371/    28BC :                      ;
6372/    28BC :                      ; Inputs:  Parent:    BYTE {R1}
6373/    28BC :                      ;          BufferPtr: PTR {R2:3}
6374/    28BC :                      ;
6375/    28BC :                      ; Outputs: ServoLoad: BOOLEAN {zero flag, true if transmission failed}
6376/    28BC :                      ;
6377/    28BC :                      ; Global Variables Changed: Buffer pointed to by RR2
6378/    28BC :                      ;
6379/    28BC :                      ; Local Variables: i:          BYTE {R4}
6380/    28BC :                      ;                  TempBufPtr: PTR {RR10}
6381/    28BC :                      ;
6382/    28BC :                      ; Algorithm:
6383/    28BC :                      ;
6384/    28BC :                      ;  Begin
6385/    28BC :                      ;    For i:=1 To 5 Do
6386/    28BC :                      ;      While not(IRQ.Serial_Input) Do Begin End
6387/    28BC :                      ;      Buffer[i]:=Sio.Data
6388/    28BC :                      ;    ServoLoad:=Check_Check_Byte(Ptr(Buffer),5)
6389/    28BC :                      ;  End
6390/    28BC :                      ;
6391/    28BC :                      ;*****************************************************************
6392/    28BC :
6393/    28BC : 70 FD                ServoLoad       push RP                 ; save context
6394/    28BE : 31 40                                srp #Wrk_Scr            ; context switch
6395/    28C0 :                              assume RP:Wrk_Scr
6396/    28C0 : EC 14                                ld R14, #SStatus0 /256
6397/    28C2 : FC B6                                ld R15, #SStatus0 #256
6398/    28C4 : 4C 05                                ld R4, #5               ; load i
6399/    28C6 :                      ;
6400/    28C6 : 76 FA 08             Srvo_Ld_Wt      tm IRQ, #Serial_In
6401/    28C9 : 6B FB                                jr Z, Srvo_Ld_Wt
6402/    28CB : 56 FA F7                             and IRQ, #0FFh-Serial_In        ; clear old interrupts
6403/    28CE : 08 F0                                ld R0, SIO              ; read SIO
6404/    28D0 : 92 0E                                lde @RR14, R0           ; and store in buffer
6405/    28D2 : A0 EE                                incw RR14               ; point to next location in buffer
6406/    28D4 : 4A F0                                djnz R4, Srvo_Ld_Wt     ; loop till all bytes are read
6407/    28D6 :                      ;
6408/    28D6 : EC 14                                ld R14, #SStatus0 /256  ; get original buffer ptr
6409/    28D8 : FC B6                                ld R15, #SStatus0 #256
6410/    28DA : 8C 04                                ld R8, #4               ; length of buffer
6411/    28DC : D6 06 1D                             call Chk_Chk_Byte
6412/    28DF : 50 FD                                pop RP                  ; context switch
6413/    28E1 : AF                                   ret
6414/    28E2 :
6415/    28E2 :
6416/    28E2 :                      ;*****************************************************************
6417/    28E2 :                      ;
6418/    28E2 :                      ; Function: Restore
6419/    28E2 :                      ;
6420/    28E2 :                      ;  This function performs a Recal operation on the servo.
6421/    28E2 :                      ;  Normally, a ReadHeader operation is to be performed after
6422/    28E2 :                      ;  the Servo comes ready, and the current cylinder is to be
6423/    28E2 :                      ;  subsequently updated. This mechanism (doing the ReadHeader)
6424/    28E2 :                      ;  can be turned off by clearing the RdHdrRecal bit in the
6425/    28E2 :                      ;  exception working register set.
6426/    28E2 :                      ;
6427/    28E2 :                      ; Inputs:  RecalType: BYTE {R0}
6428/    28E2 :                      ;
6429/    28E2 :                      ; Outputs: Restore: BOOLEAN {zero flag, true if not(ServoRdy)}
6430/    28E2 :                      ;
6431/    28E2 :                      ; Algorithm:
6432/    28E2 :                      ;
6433/    28E2 :                      ;  Begin
6434/    28E2 :                      ;    ServoCmnd(RecalType, 0, 0, 57.6kBaud)
6435/    28E2 :                      ;    While not(ServoRdy) And 2 seconds hasn't gone by Do Begin End
6436/    28E2 :                      ;    If DiskStat.RdHdrRecal and ServoRdy
6437/    28E2 :                      ;      Then If not(UpDate_Cur_Cyl) Then Abort
6438/    28E2 :                      ;      Else
6439/    28E2 :                      ;        If RecalType=DataRecal
6440/    28E2 :                      ;          Then CurCyl:=Init_Cyl
6441/    28E2 :                      ;          Else CurCyl:=Max_Cyl
6442/    28E2 :                      ;    Restore:=ServoRdy
6443/    28E2 :                      ;  End
6444/    28E2 :                      ;
6445/    28E2 :                      ;*****************************************************************
6446/    28E2 :
6447/    28E2 : 70 E0                Restore         push R0                 ; save Recal type for later
6448/    28E4 : D6 22 15                             call Ext_Push
6449/    28E7 : D6 04 13                             call Set_Dmt
6450/    28EA : 31 40                                srp #Wrk_Scr
6451/    28EC :                              assume RP:Wrk_Scr
6452/    28EC : 50 E0                                pop R0                  ; get back Recal type
6453/    28EE : 70 E0                                push R0
6454/    28F0 : B0 E1                                clr R1
6455/    28F2 : B0 E2                                clr R2
6456/    28F4 : 3C 80                                ld R3, #S_Rate_57_6
6457/    28F6 : 31 10                                srp #Wrk_Sys
6458/    28F8 :                              assume RP:Wrk_Sys
6459/    28F8 : D6 28 37                             call ServoCmnd
6460/    28FB : D6 04 1F                             call Clr_Dmt
6461/    28FE : 56 56 7F                             and DiskStat, #0FFh-On_Track
6462/    2901 : D6 2A 0E                             call Set_SeekNeeded     ; invalidate cache contents
6463/    2904 : E6 50 02                             ld Cur_Cyl, #HiMaxCyl   ; assume FrmtRecal
6464/    2907 : E6 51 20                             ld Cur_Cyl+1, #LoMaxCyl
6465/    290A : 50 E6                                pop R6                  ; load Recal type from storage
6466/    290C : A6 E6 70                             cp R6, #FrmtRecal
```

```
6467/    290F : 6B 06                            jr Z, Rest_Up2
6468/    2911 : E6 50 01                         ld Cur_Cyl, #Init_HiCyl ; otherwise DataRecal
6469/    2914 : E6 51 F9                         ld Cur_Cyl+1, #Init_LoCyl
6470/    2917 : B0 E5               Rest_Up2     clr R5
6471/    2919 : 4C D5                            ld R4, #0D5h            ; max time to wait is two seconds
6472/    291B :                     ;
6473/    291B : D6 03 FC            Restore_Lp   call LoadStatus
6474/    291E : 76 E0 20                         tm R0, #ServoRdy
6475/    2921 : EB 08                            jr NZ, Rest_UpDate
6476/    2923 : 80 E4                            decw RR4
6477/    2925 : EB F4                            jr NZ, Restore_Lp
6478/    2927 : B0 E0                            clr R0                  ; pass error status to exit
6479/    2929 : 8B 17                            jr Restore_End
6480/    292B :                     ;
6481/    292B : 76 56 40            Rest_UpDate  tm DiskStat, #RdHdrRecal
6482/    292E : 6B 10                            jr Z, Rest_Up1
6483/    2930 : A6 E6 70                         cp R6, #FrmtRecal       ; don't try to read headers here!
6484/    2933 : 6B 0B                            jr Z, Rest_Up1
6485/    2935 : D6 20 EB                         call UpDate_Cur_Cyl
6486/    2938 : EB 06                            jr NZ, Rest_Up1         ; leave if positioned correctly
6487/    293A :                     ;
6488/    293A : D6 05 A2                         call SS_NoHdr           ; header err here is bad news!
6489/    293D : D6 05 1F                         call Abort
6490/    2940 :                     ;
6491/    2940 : 0C 01               Rest_Up1     ld R0, #1
6492/    2942 : 70 E0               Restore_End  push R0                 ; save result
6493/    2944 : D6 22 36                         call Ext_Pop
6494/    2947 : 56 56 6E                         and DiskStat, #0FFh-Offset_On-Parked-On_Track
6495/    294A : 46 56 02                         or DiskStat, #SeekComplete
6496/    294D : 50 E0                            pop R0
6497/    294F : 42 00                            or R0, R0               ; set zero flag
6498/    2951 : 8D 04 F8                         jp Bank_Ret
6499/    2954 :
6500/    2954 :
6501/    2954 :                     ;*********************************************************************
6502/    2954 :                     ;
6503/    2954 :                     ; Procedure: SrvoRcvry  {servo recovery}
6504/    2954 :                     ;
6505/    2954 :                     ;  This procedure's responsibility is to do everything within reason
6506/    2954 :                     ;  to make certain that the Servo Processor is Healthy, Wealthy, and
6507/    2954 :                     ;  Wise; and if it can't then there is no point in using the Servo.
6508/    2954 :                     ;
6509/    2954 :                     ; Inputs:  none
6510/    2954 :                     ;
6511/    2954 :                     ; Outputs: none
6512/    2954 :                     ;
6513/    2954 :                     ; Local Variables: i:     BYTE {R8}
6514/    2954 :                     ;                  Error: BOOLEAN {R9}
6515/    2954 :                     ;
6516/    2954 :                     ; Global Variabled Changed: On_Track, Cur_Cyl
6517/    2954 :                     ;
6518/    2954 :                     ; Global Variables Used: Cylinder, Head, Sector
6519/    2954 :                     ;
6520/    2954 :                     ; Algorithm:
6521/    2954 :                     ;
6522/    2954 :                     ;  Begin
6523/    2954 :                     ;    i:=4
6524/    2954 :                     ;    Repeat
6525/    2954 :                     ;      ZeroHeader
6526/    2954 :                     ;      Error:=false
6527/    2954 :                     ;      If ServoError
6528/    2954 :                     ;        Then Error:=not(ServoOk)
6529/    2954 :                     ;        Else
6530/    2954 :                     ;         Error:=not(ReadHdr)
6531/    2954 :                     ;         If not(Error)
6532/    2954 :                     ;          Then
6533/    2954 :                     ;           If ReadHdr.Cylinder<>Cylinder
6534/    2954 :                     ;            Then
6535/    2954 :                     ;             Cur_Cyl:=ReadHdr.Cylinder
6536/    2954 :                     ;             On_Track:=false
6537/    2954 :                     ;      If not(On_Track) Then Seek(Cylinder, Head, Sector)
6538/    2954 :                     ;    Until i=0 Or not(Error)
6539/    2954 :                     ;    If i=0 Then Abort
6540/    2954 :                     ;  End
6541/    2954 :                     ;
6542/    2954 :                     ;*********************************************************************
6543/    2954 :
6544/    2954 : D6 22 15            SrvoRcvry    call Ext_Push           ; save caller's stuff
6545/    2957 : 8C 04                            ld R8, #4               ; i:=4
6546/    2959 :                     ;
6547/    2959 : D6 22 8B            Srvo_R_Rpt   call ZeroHeader
6548/    295C : D6 27 08                         call Chk_SStat
6549/    295F : EB 03                            jr NZ, Srvo_R_Sok
6550/    2961 : D6 27 5A                         call ServoOk
6551/    2964 :                     ;
6552/    2964 : D6 21 66            Srvo_R_Sok   call UpDate_Hdr         ; If ReadHdr.Cylinder<>Cylinder
6553/    2967 : EB 0C                            jr NZ, Srvo_R_Leave
6554/    2969 : 0C 01                            ld R0, #1               ; status byte 1
6555/    296B : 1C 02                            ld R1, #Stat_Seek
6556/    296D : D6 04 03                         call SetStatus
6557/    2970 : D6 05 CA                         call ReSeek
6558/    2973 : 8A E4                            djnz R8, Srvo_R_Rpt
6559/    2975 :                     ;
6560/    2975 : D6 22 36            Srvo_R_Leave call Ext_Pop            ; get user's stuff back
6561/    2978 : 8D 04 F8                         jp Bank_Ret
6562/    297B :
6563/    297B :
6564/    297B :                     ;*********************************************************************
6565/    297B :                     ;
6566/    297B :                     ; Procedure: Load_SrvoCmnd  {load servo command buffer}
6567/    297B :                     ;
6568/    297B :                     ;  This procedure loads the global array ServoCmndBuffer
6569/    297B :                     ;  with the information contained within R0:3
6570/    297B :                     ;
6571/    297B :                     ; Inputs: SrvoCmndBuffer[0]: BYTE {R0}
6572/    297B :                     ;         SrvoCmndBuffer[1]: BYTE {R1}
6573/    297B :                     ;         SrvoCmndBuffer[2]: BYTE {R2}
6574/    297B :                     ;         SrvoCmndBuffer[3]: BYTE {R3}
6575/    297B :                     ;
6576/    297B :                     ; Outputs: none
6577/    297B :                     ;
6578/    297B :                     ;*********************************************************************
6579/    297B :
6580/    297B : 2C 14               Load_SrvoCmnd ld R2, #SrvoCmndBuf /256
6581/    297D : 3C B1                             ld R3, #SrvoCmndBuf #256
6582/    297F : 1C 04                             ld R1, #4               ; load 4 bytes
6583/    2981 : 0C 40                             ld R0, #ScrReg0         ; start with what's in ScrReg0
6584/    2983 : 93 02               Ld_S_Cmnd_Lp  ldei @RR2, @R0
```

```
6585/    2985 : 1A FC                            djnz R1, Ld_S_Cmnd_Lp
6586/    2987 : AF                               ret
6587/    2988 :
6588/    2988 :
6589/    2988 :                     ;*********************************************************************
6590/    2988 :                     ;
6591/    2988 :                     ; Procedure: Park_Heads
6592/    2988 :                     ;
6593/    2988 :                     ;  This procedure moves the heads in a closed-loop fashion
6594/    2988 :                     ;  (access command vs Park command) to a location away from the
6595/    2988 :                     ;  user data area. The attempt here is to provide some additional
6596/    2988 :                     ;  protection from power failures, and the such, from accidentally
6597/    2988 :                     ;  writing bogus data on the disk.
6598/    2988 :                     ;
6599/    2988 :                     ; Inputs: none
6600/    2988 :                     ;
6601/    2988 :                     ; Outputs: none
6602/    2988 :                     ;
6603/    2988 :                     ; Algorithm:
6604/    2988 :                     ;
6605/    2988 :                     ;  Begin
6606/    2988 :                     ;    Seek(HiParkCylinder, LowParkCylinder, 0, 0)
6607/    2988 :                     ;    Set_SeekNeeded
6608/    2988 :                     ;  End
6609/    2988 :                     ;
6610/    2988 :                     ;*********************************************************************
6611/    2988 :
6612/    2988 : E6 57 80            Park_Heads      ld Seek_Type, #Access
6613/    298B : CC 02                               ld R12, #ParkCyl /256
6614/    298D : DC 35                               ld R13, #ParkCyl #256
6615/    298F : B0 EE                               clr R14
6616/    2991 : B0 EF                               clr R15
6617/    2993 : D6 26 05                            call Seek
6618/    2996 : 46 56 10                            or DiskStat, #Parked
6619/    2999 : 56 56 7F                            and DiskStat, #0FFh-On_Track
6620/    299C : D6 2A 0E                            call Set_SeekNeeded     ; invalidate cache contents
6621/    299F : 8D 04 F8                            jp Bank_Ret
6622/    29A2 :
6623/    29A2 :
6624/    29A2 :                     ;*********************************************************************
6625/    29A2 :                     ;
6626/    29A2 :                     ; Procedure: ResetServo
6627/    29A2 :                     ;
6628/    29A2 :                     ;  This procedure is responsible for performing all the
6629/    29A2 :                     ;  necessary tasks in resetting the servo. This includes
6630/    29A2 :                     ;  setting the correct Baud rates (the servo comes up
6631/    29A2 :                     ;  at 19.2k Baud, but normally runs at 57.6k Baud) and
6632/    29A2 :                     ;  positioning the heads over the data field and keeping
6633/    29A2 :                     ;  the world straight about it (cylinder is set to
6634/    29A2 :                     ;  MaxDataCylinder because a DataRecal positions the heads
6635/    29A2 :                     ;  at the closest data cylinder to the inside)
6636/    29A2 :                     ;
6637/    29A2 :                     ; Inputs: none
6638/    29A2 :                     ;
6639/    29A2 :                     ; Outputs: none
6640/    29A2 :                     ;
6641/    29A2 :                     ; Global Variables Changed: Cylinder
6642/    29A2 :                     ;
6643/    29A2 :                     ; Algorithm:
6644/    29A2 :                     ;
6645/    29A2 :                     ;  Begin
6646/    29A2 :                     ;    ServoRst:=true
6647/    29A2 :                     ;    wait for 50ms to make certain that Reset is valid
6648/    29A2 :                     ;    ServoRst:=false
6649/    29A2 :                     ;    wait for 1 sec to allow motor and servo to get ready
6650/    29A2 :                     ;    If not(ServoRdy) or ServoError Then Abort
6651/    29A2 :                     ;    Baud Rate:=19.2k
6652/    29A2 :                     ;    If not(ServoStore(ReadStatus, x, x, BaudRate57.6+NormalStatus)
6653/    29A2 :                     ;      Then Abort
6654/    29A2 :                     ;    If not(ServoLoad(NormalReadStatusBuffer)
6655/    29A2 :                     ;      Then Abort
6656/    29A2 :                     ;    Baud Rate:=57.6k
6657/    29A2 :                     ;    If not(Restore(DataRecal)) Then Abort
6658/    29A2 :                     ;    DiskStatus.On_Track:=false
6659/    29A2 :                     ;  End
6660/    29A2 :                     ;
6661/    29A2 :                     ;*********************************************************************
6662/    29A2 :
6663/    29A2 : 56 00 EF            ResetServo      and P0, #0FFh-Not_ServoRst      ; set Servo Reset
6664/    29A5 : 2C 00                               ld R2, #5 /256
6665/    29A7 : 3C 05                               ld R3, #5 #256
6666/    29A9 : D6 01 CB                            call MsWait             ; busy wait for 50ms
6667/    29AC : 46 00 10                            or P0, #Not_ServoRst    ; clear Servo Reset
6668/    29AF : 2C 00                               ld R2, #100 /256
6669/    29B1 : 3C 64                               ld R3, #100 #256
6670/    29B3 : D6 01 CB                            call MsWait             ; busy wait for 1s
6671/    29B6 : D6 03 FC            S_Rst_1         call LoadStatus         ; get servo status
6672/    29B9 : 76 E0 10                            tm R0, #ServoErr
6673/    29BC : 6B 05                               jr Z, S_Rst_Bd1
6674/    29BE :                     ;
6675/    29BE : A8 E0               S_Rst_Abt1      ld R10, R0
6676/    29C0 : D6 05 1F                            call Abort
6677/    29C3 :                     ;
6678/    29C3 : 56 F1 FD            S_Rst_Bd1       and TMR, #0FFh-T0_CntEn ; halt T0
6679/    29C6 : E6 F5 0D                            ld PRE0, #00Dh          ; PRE0:=Mod-64, continuous, go to 19.2k
6680/    29C9 : E6 F4 01                            ld T0, #1               ; interrupt after 1 byte
6681/    29CC : 46 F1 03                            or TMR, #T0_CntEn+T0_Load
6682/    29CF : D6 04 13                            call Set_Dmt
6683/    29D2 : 31 40                               srp #Wrk_Scr
6684/    29D4 :                                     assume RP:Wrk_Scr
6685/    29D4 : 0C 00                               ld R0, #ReadStatus      ; try talking to the Servo
6686/    29D6 : B0 E1                               clr R1
6687/    29D8 : B0 E2                               clr R2
6688/    29DA : 3C 81                               ld R3, #S_Norm_Status+S_Rate_57_6
6689/    29DC : 31 10                               srp #Wrk_Sys
6690/    29DE :                                     assume RP:Wrk_Sys
6691/    29DE : D6 29 7B                            call Load_SrvoCmnd
6692/    29E1 : D6 28 7F                            call ServoStore
6693/    29E4 : 6B 08                               jr Z, S_Rst_Comm
6694/    29E6 : D6 03 FC                            call LoadStatus
6695/    29E9 : 76 E0 10                            tm R0, #ServoErr
6696/    29EC : 6B 07                               jr Z, S_Rst_Ld
6697/    29EE :                     ;
6698/    29EE : AC 00               S_Rst_Comm      ld R10, #S_Store
6699/    29F0 : 8C 03               S_Rst_Abt2      ld R8, #S_Rst_Abort
6700/    29F2 : D6 05 1F                            call Abort
6701/    29F5 :                     ;
6702/    29F5 : D6 28 BC            S_Rst_Ld        call ServoLoad
```

```
6703/    29F8 : 7C 01                           ld R7, #S_Load
6704/    29FA : EB F4                           jr NZ, S_Rst_Abt2
6705/    29FC : D6 04 1F                        call Clr_Dmt
6706/    29FF : 56 F1 FD                        and TMR, #0FFh-T0_CntEn ; halt T0
6707/    2A02 : E6 F5 05                        ld PRE0, #5             ; PRE0:=1, continuous run, go to 57.6k
6708/    2A05 : E6 F4 01                        ld T0, #1
6709/    2A08 : 46 F1 03                        or TMR, #T0_CntEn+T0_Load
6710/    2A0B : 8D 04 F8                        jp Bank_Ret
6711/    2A0E :
6712/    2A0E :
6713/    2A0E :                     ;*********************************************************************
6714/    2A0E :                     ;
6715/    2A0E :                     ; Procedure: Set_SeekNeeded
6716/    2A0E :                     ;
6717/    2A0E :                     ;  This procedure acts in much the same fashion for the
6718/    2A0E :                     ;  LogicalBlockCache as DiskStat.On_Track does for simple
6719/    2A0E :                     ;  seeks. It keeps a legitimate seeks between two requests
6720/    2A0E :                     ;  for the same block number from returning a bogus result.
6721/    2A0E :                     ;
6722/    2A0E :                     ; Inputs: none
6723/    2A0E :                     ;
6724/    2A0E :                     ; Outputs: none
6725/    2A0E :                     ;
6726/    2A0E :                     ; Global Variables Changed: Cylinder
6727/    2A0E :                     ;
6728/    2A0E :                     ; Algorithm:
6729/    2A0E :                     ;
6730/    2A0E :                     ;  Begin
6731/    2A0E :                     ;    For i:=1 To CacheLength Do
6732/    2A0E :                     ;      CacheStat[i].SeekNeeded:=true
6733/    2A0E :                     ;  End
6734/    2A0E :                     ;
6735/    2A0E :                     ;*********************************************************************
6736/    2A0E :
6737/    2A0E : 2C 16               Set_SeekNeeded  ld R2, #CacheStat /256
6738/    2A10 : 3C E8                               ld R3, #CacheStat #256
6739/    2A12 : 1C 14                               ld R1, #CacheLength
6740/    2A14 : 82 02               S_SeekN_Lp      lde R0, @RR2            ; get array value
6741/    2A16 : 46 E0 80                            or R0, #CachSeek
6742/    2A19 : 92 02                               lde @RR2, R0
6743/    2A1B : A0 E2                               incw RR2
6744/    2A1D : 1A F5                               djnz R1, S_SeekN_Lp
6745/    2A1F : 8D 04 F8                            jp Bank_Ret
6746/    2A22 :
6747/    2A22 :
6748/    2A22 :
6749/    2A22 :                     ;*********************************************************************
6750/    2A22 :                     ; Module SlfTst.B1.Assem   {Selftest}
6751/    2A22 :                     ;
6752/    2A22 :                     ;       FUNCTION: MtrSpd
6753/    2A22 :                     ;       FUNCTION: TrackCount
6754/    2A22 :                     ;       FUNCTION: SctrCount
6755/    2A22 :                     ;       FUNCTION: RwTest
6756/    2A22 :                     ;
6757/    2A22 :                     ;*********************************************************************
6758/    2A22 :
6759/    2A22 :                     ;*********************************************************************
6760/    2A22 :                     ;
6761/    2A22 :                     ; Function: Selftest
6762/    2A22 :                     ;
6763/    2A22 :                     ;  This function performs some check to confirm Widget's integrity.
6764/    2A22 :                     ;
6765/    2A22 :                     ;  Inputs: none
6766/    2A22 :                     ;
6767/    2A22 :                     ;  Outputs: SlfTst_Result: BYTE
6768/    2A22 :                     ;
6769/    2A22 :                     ;*********************************************************************
6770/    2A22 :
6771/    2A22 : 46 24 80            SelfTest        or Excpt_Stat, #Recovery        ; try our best to pass this test
6772/    2A25 : 76 24 10                            tm Excpt_Stat, #PwrRst  ; check for power reset
6773/    2A28 : 6B 0E                               jr Z, ST_MtrSpd
6774/    2A2A :                     ;
6775/    2A2A : 08 38                               ld R0, Scr_Cntr         ; otherwise finish time on disk speed
6776/    2A2C : 44 39 E0                            or R0, Scr_Cntr+1
6777/    2A2F : 6B 07                               jr Z, ST_MtrSpd
6778/    2A31 :                     ;
6779/    2A31 : 28 38                               ld R2, Scr_Cntr
6780/    2A33 : 38 39                               ld R3, Scr_Cntr+1
6781/    2A35 : D6 01 CB                            call MsWait
6782/    2A38 :                     ;
6783/    2A38 : D6 2A 6D            ST_MtrSpd       call MtrSpd
6784/    2A3B : 6B 09                               jr Z, ST_Mtr_Gd         ; ok, passed
6785/    2A3D : 2C 05                               ld R2, #1500 /256
6786/    2A3F : 3C DC                               ld R3, #1500 #256       ; wait another 21 ms
6787/    2A41 : D6 2A 6D                            call MtrSpd             ;  (label should be MtrSpd_Lp1 !)
6788/    2A44 : EB 24                               jr NZ, Slf_Leave
6789/    2A46 :                     ;
6790/    2A46 : 56 25 DF            ST_Mtr_Gd       and SlfTst_Result, #0FFh-Disk_Speed
6791/    2A49 : D6 2A B1            Slf_Tracks      call TrackCount
6792/    2A4C : 56 25 EF                            and SlfTst_Result, #0FFh-Servo_Fail
6793/    2A4F : D6 2A C8            Slf_Sectors     call SctrCount
6794/    2A52 : EB 16                               jr NZ, Slf_Leave
6795/    2A54 : 56 25 F7                            and SlfTst_Result, #0FFh-Sector_Cnt
6796/    2A57 : D6 03 FC            Slf_State       call LoadStatus         ; check state machine state
6797/    2A5A : 56 E0 0F                            and R0, #YMask
6798/    2A5D : EB 0B                               jr NZ, Slf_Leave
6799/    2A5F : 56 25 FB                            and SlfTst_Result, #0FFh-State_Fail
6800/    2A62 : D6 2A E9            Slf_RwTest      call RwTest             ; see if we can read and write
6801/    2A65 : 6B 03                               jr Z, Slf_Leave
6802/    2A67 : 56 25 FD                            and SlfTst_Result, #0FFh-Rw_Fail
6803/    2A6A :                     ;
6804/    2A6A : 8D 04 F8            Slf_Leave       jp Bank_Ret
6805/    2A6D :
6806/    2A6D :
6807/    2A6D :                     ;*********************************************************************
6808/    2A6D :                     ;
6809/    2A6D :                     ; Function: MtrSpd  {motor speed}
6810/    2A6D :                     ;
6811/    2A6D :                     ;  This function is responsible for measuring the motor
6812/    2A6D :                     ;  speed. This is done by timing the interval between
6813/    2A6D :                     ;  consecutive index marks
6814/    2A6D :                     ;
6815/    2A6D :                     ; Inputs: none
6816/    2A6D :                     ;
6817/    2A6D :                     ; Outputs: MtrSpd: BOOLEAN {zero flag, NOT set if failure}
6818/    2A6D :                     ;
6819/    2A6D :                     ;*********************************************************************
6820/    2A6D :
```

```
6821/    2A6D : 56 FA FE            MtrSpd          and IRQ, #0FFh-IRQ_Index        ; clear old event
6822/    2A70 : 0C 01                               ld R0, #1               ; assume failure
6823/    2A72 : 4C 06                               ld R4, #1600 /256       ; load dead man timer
6824/    2A74 : 5C 40                               ld R5, #1600 #256       ;  (22,7ms)
6825/    2A76 : 2C 2A                               ld R2, #11000 /256      ; timeout after about 4 rotations
6826/    2A78 : 3C F8                               ld R3, #11000 #256      ;  (156,2ms)
6827/    2A7A : 76 FA 01            MtrSpd_Lp1      tm IRQ, #Irq_Index      ; wait for index to come around
6828/    2A7D : EB 06                               jr NZ, MtrSpd_Idx       ;  got him
6829/    2A7F : 80 E2                               decw RR2                ; decrement timeout
6830/    2A81 : EB F7                               jr NZ, MtrSpd_Lp1
6831/    2A83 : 8B 29                               jr MtrSpd_Lv
6832/    2A85 :                     ;
6833/    2A85 : 56 FA FA            MtrSpd_Idx      and IRQ, #0FFh-IRQ_Index-Irq_Sector     ; clear events
6834/    2A88 : B0 E0                               clr R0                  ; clear counter
6835/    2A8A : B0 E1                               clr R1
6836/    2A8C : 76 FA 01            MtrSpd_Lp2      tm IRQ, #Irq_Index      ; wait for one rev
6837/    2A8F : EB 06                               jr NZ, MtrSpd_End       ;  done
6838/    2A91 : A0 E0                               incw RR0                ; 0.0142 ms/lp or 70.4 cnts/ms
6839/    2A93 : 80 E4                               decw RR4                ; decrement timeout
6840/    2A95 : EB F5                               jr NZ, MtrSpd_Lp2
6841/    2A97 :                     ; nominal value is 1408 counts for 20 ms / 3000 rpm
6842/    2A97 : E8 E1               MtrSpd_End      ld R14, R1
6843/    2A99 : D8 E0                               ld R13, R0
6844/    2A9B : CC 00                               ld R12, #0
6845/    2A9D : D6 25 7B                            call Div3_19            ; get speed within tolerance in R2
6846/    2AA0 : =>TRUE                      if W_10MB
6847/    2AA0 : 0C 01                               ld R0, #1               ; assume failure
6848/    2AA2 : A6 E2 44                            cp R2, #044h            ; accept speeds between 18.4 and 20.5 ms
6849/    2AA5 : 1B 07                               jr LT, MtrSpd_Lv
6850/    2AA7 : A6 E2 4C                            cp R2, #4Ch
6851/    2AAA : AB 02                               jr GT, MtrSpd_Lv
6852/    2AAC : 0C 00                               ld R0, #0               ; otherwise pass
6853/    2AAE : [6846]                      endif
6854/    2AAE : =>FALSE                     if W_20MB || W_40MB
6855/    2AAE :                                     ld R0, #1               ; assume failure
6856/    2AAE :                     ; 1A44          cp R2, #044h            ; accept speeds between 18.5 and 20.5 ms
6857/    2AAE :                                     cp R2, #048h            ; accept speeds between 19.5 and 20.5 ms
6858/    2AAE :                                     jr LT, MtrSpd_Lv
6859/    2AAE :                                     cp R2, #4Ch
6860/    2AAE :                                     jr GT, MtrSpd_Lv
6861/    2AAE :                                     ld R0, #0               ; otherwise pass
6862/    2AAE : [6854]                      endif
6863/    2AAE :                     ;
6864/    2AAE : 42 00               MtrSpd_Lv       or R0, R0               ; set flags
6865/    2AB0 : AF                                  ret
6866/    2AB1 :
6867/    2AB1 :
6868/    2AB1 :                     ;*********************************************************************
6869/    2AB1 :                     ;
6870/    2AB1 :                     ; Function: TrackCount
6871/    2AB1 :                     ;
6872/    2AB1 :                     ;  This function is responsible for counting the number of
6873/    2AB1 :                     ;  tracks that are available on the servo.
6874/    2AB1 :                     ;
6875/    2AB1 :                     ; Inputs: none
6876/    2AB1 :                     ;
6877/    2AB1 :                     ; Outputs: TrackCount: BOOLEAN {zero flag, NOT set if failure
6878/    2AB1 :                     ;
6879/    2AB1 :                     ; Algorithm:
6880/    2AB1 :                     ;
6881/    2AB1 :                     ;  Begin
6882/    2AB1 :                     ;   Restore(FrmtRecal)
6883/    2AB1 :                     ;   Seek(Cyl=0)
6884/    2AB1 :                     ;  End
6885/    2AB1 :                     ;
6886/    2AB1 :                     ;*********************************************************************
6887/    2AB1 :
6888/    2AB1 : D6 29 A2            TrackCount      call ResetServo         ; try to get the servo in a nice state
6889/    2AB4 : 56 56 BF                            and DiskStat, #0FFh-RdHdrRecal  ; don't read any headers!
6890/    2AB7 : 0C 40                               ld R0, #DataRecal
6891/    2AB9 : D6 28 E2                            call Restore
6892/    2ABC : CC 00                               ld R12, #0              ; cylinder = 0
6893/    2ABE : DC 69                               ld R13, #69h
6894/    2AC0 : EC 00                               ld R14, #0              ; head = 0
6895/    2AC2 : FC 00                               ld R15, #0              ; sector = 0
6896/    2AC4 : D6 26 05                            call Seek
6897/    2AC7 : AF                                  ret
6898/    2AC8 :
6899/    2AC8 :
6900/    2AC8 :                     ;*********************************************************************
6901/    2AC8 :                     ;
6902/    2AC8 :                     ; Function: SctrCount
6903/    2AC8 :                     ;
6904/    2AC8 :                     ;  This function counts the number of sector marks between
6905/    2AC8 :                     ;  successive index marks and returns a true value if
6906/    2AC8 :                     ;  the number of sectors counted equals NbrSctrs.
6907/    2AC8 :                     ;
6908/    2AC8 :                     ; Inputs: none
6909/    2AC8 :                     ;
6910/    2AC8 :                     ; Outputs: SctrCount: BOOLEAN {zero flag, NOT set if failure}
6911/    2AC8 :                     ;
6912/    2AC8 :                     ;*********************************************************************
6913/    2AC8 :
6914/    2AC8 : B0 E2               SctrCount       clr R2                  ; count:=0
6915/    2ACA : 56 FA FE                            and IRQ, #0FFh-Irq_Index        ; clear old index marks
6916/    2ACD : 76 FA 01            S_Cnt_1         tm IRQ, #Irq_Index      ; wait for index mark
6917/    2AD0 : 6B FB                               jr Z, S_Cnt_1
6918/    2AD2 : 56 FA FD                            and IRQ, #0FFh-Irq_SecDn        ; clear old events
6919/    2AD5 : 76 03 04            S_Cnt_3         tm P3, #IndexMark       ; While not(Index)
6920/    2AD8 : EB 0B                               jr NZ, S_Cnt_End
6921/    2ADA : 76 FA 04                            tm IRQ, #Irq_Sector
6922/    2ADD : 6B F6                               jr Z, S_Cnt_3
6923/    2ADF : 2E                                  inc R2                  ; bump the sector count
6924/    2AE0 : 56 FA FB                            and IRQ, #0FFh-Irq_Sector
6925/    2AE3 : 8B F0                               jr S_Cnt_3
6926/    2AE5 : A6 E2 13            S_Cnt_End       cp R2, #NbrSctrs
6927/    2AE8 : AF                                  ret
6928/    2AE9 :
6929/    2AE9 :
6930/    2AE9 :                     ;*********************************************************************
6931/    2AE9 :                     ;
6932/    2AE9 :                     ; Function: RwTest
6933/    2AE9 :                     ;
6934/    2AE9 :                     ;  This test moves the heads over a track away from the data
6935/    2AE9 :                     ;  area (useable data area) and performs a write verify
6936/    2AE9 :                     ;  on block zero of that track. If a failure occurs, the test
6937/    2AE9 :                     ;  will continue on the next sequential sector on that track
6938/    2AE9 :                     ;  until all sectors have been written to. If there are no
```

```
6939/    2AE9 :                     ;  successful transfers on any of the sectors of this track then
6940/    2AE9 :                     ;  the test is assumed to have failed.
6941/    2AE9 :                     ;
6942/    2AE9 :                     ; Inputs: none
6943/    2AE9 :                     ;
6944/    2AE9 :                     ; Outputs: RwTest: BOOLEAN {zero flag, NOT set if failure
6945/    2AE9 :                     ;
6946/    2AE9 :                     ; Algorithm:
6947/    2AE9 :                     ;
6948/    2AE9 :                     ;  Begin
6949/    2AE9 :                     ;    Seek(RwTest track)
6950/    2AE9 :                     ;    Repeat
6951/    2AE9 :                     ;      If not(WrVerCommon) And RdErrCnt=10
6952/    2AE9 :                     ;        Then
6953/    2AE9 :                     ;          Sector:=Sector+1
6954/    2AE9 :                     ;          If Sector>NbrSctrs
6955/    2AE9 :                     ;            Then Done:=true
6956/    2AE9 :                     ;       Else Done:=true
6957/    2AE9 :                     ;    Until Done
6958/    2AE9 :                     ;    If Sector>NbrSctrs
6959/    2AE9 :                     ;      Then RwTest:=false
6960/    2AE9 :                     ;      Else RwTest:=true
6961/    2AE9 :                     ;  End
6962/    2AE9 :                     ;
6963/    2AE9 :                     ;*****************************************************************
6964/    2AE9 :
6965/    2AE9 : 46 56 40            RwTest          or DiskStat, #RdHdrRecal
6966/    2AEC : 0C 40                               ld R0, #DataRecal
6967/    2AEE : D6 28 E2                            call Restore
6968/    2AF1 : 5C 00                               ld R5, #0
6969/    2AF3 : 6B 46                               jr Z, RwTest_End
6970/    2AF5 :                     ;
6971/    2AF5 : CC 02               RwTest1         ld R12, #Tst_HiCyl
6972/    2AF7 : DC 05                               ld R13, #Tst_LoCyl
6973/    2AF9 : EC 00                               ld R14, #Tst_Head
6974/    2AFB : FC 00                               ld R15, #Tst_Sctr
6975/    2AFD : D6 26 05                            call Seek
6976/    2B00 : 5C 13                               ld R5, #NbrSctrs
6977/    2B02 :                     ;
6978/    2B02 : D6 22 8B            RwTest_Lp       call ZeroHeader
6979/    2B05 : 70 E5                               push R5
6980/    2B07 : 46 56 20                            or DiskStat, #Wr_Op
6981/    2B0A : 2C 10                               ld R2, #WrBlk_Vector /256
6982/    2B0C : 3C 7D                               ld R3, #WrBlk_Vector #256
6983/    2B0E : D6 04 AB                            call Bank_Call
6984/    2B11 : 50 E5                               pop R5
6985/    2B13 : 6B 22                               jr Z, Rw_Next
6986/    2B15 :                     ;
6987/    2B15 : E6 27 0A                            ld RdErrCnt, #10        ; assume a failure
6988/    2B18 : 70 E5                               push R5
6989/    2B1A : 56 56 DF                            and DiskStat, #0FFh-Wr_Op
6990/    2B1D : 2C 10                               ld R2, #RdBlk_Vector /256
6991/    2B1F : 3C 83                               ld R3, #RdBlk_Vector #256
6992/    2B21 : D6 04 AB                            call Bank_Call
6993/    2B24 : 50 E5                               pop R5
6994/    2B26 : EB 13                               jr NZ, RwTest_End
6995/    2B28 :                     ;
6996/    2B28 : 76 E0 10                            tm R0, #RdNoHdrFnd
6997/    2B2B : EB 0A                               jr NZ, Rw_Next
6998/    2B2D : 08 27                               ld R0, RdErrCnt
6999/    2B2F : 56 E0 0F                            and R0, #0Fh            ; mask off unwanted status
7000/    2B32 : A6 E0 0A                            cp R0, #10
7001/    2B35 : EB 04                               jr NZ, RwTest_End
7002/    2B37 :                     ;
7003/    2B37 : 20 55               Rw_Next         inc Sector
7004/    2B39 : 5A C7                               djnz R5, RwTest_Lp
7005/    2B3B : D6 29 88            RwTest_End      call Park_Heads
7006/    2B3E : 42 55                               or R5, R5               ; set zero flag
7007/    2B40 : AF                                  ret
7008/    2B41 :
7009/    2B41 :
7010/    2B41 :
7011/    2B41 :                     ;*****************************************************************
7012/    2B41 :                     ; Module Cache.B1.Assem
7013/    2B41 :                     ;
7014/    2B41 :                     ; This module contains those Cache routines that must be located
7015/    2B41 :                     ; in Bank 1.
7016/    2B41 :                     ;
7017/    2B41 :                     ;       PROCEDURE Load_Cache
7018/    2B41 :                     ;
7019/    2B41 :                     ;*****************************************************************
7020/    2B41 :
7021/    2B41 :                     ;*****************************************************************
7022/    2B41 :                     ;
7023/    2B41 :                     ; Procedure: Load_Cache
7024/    2B41 :                     ;
7025/    2B41 :                     ;  This procedure loads the 'cache-table' with the next 20
7026/    2B41 :                     ;  sequential logical blocks (from the current logical block)
7027/    2B41 :                     ;  and status' associated with each block (i.e. if no seek is
7028/    2B41 :                     ;  required, if a head change is required, etc) and the
7029/    2B41 :                     ;  head and sector number of each logical block (assuming that
7030/    2B41 :                     ;  all the blocks are on the same track). It should be noted that
7031/    2B41 :                     ;  the status bytes are kept in a separate table.
7032/    2B41 :                     ;
7033/    2B41 :                     ; Inputs: none
7034/    2B41 :                     ;
7035/    2B41 :                     ; Outputs: none
7036/    2B41 :                     ;
7037/    2B41 :                     ; Algorithm:
7038/    2B41 :                     ;
7039/    2B41 :                     ;  Begin
7040/    2B41 :                     ;    Data_Type:=User_Type
7041/    2B41 :                     ;    For i:=1 To CacheLength Do
7042/    2B41 :                     ;      Cur_Block:=LogicalBlock+i
7043/    2B41 :                     ;      TempCyl, TempHead, TempSector,
7044/    2B41 :                     ;          SrchStat, SrchPtr:=CnvrtLogical(Cur_Block)
7045/    2B41 :                     ;      CacheStatus[i-1].Seek:=CalcMagDir(TempCyl)
7046/    2B41 :                     ;      If TempHead<>Head
7047/    2B41 :                     ;        Then CacheStatus[i.1].HeadChange:=True
7048/    2B41 :                     ;        Else CacheStatus[i-1].HeadChange:=false
7049/    2B41 :                     ;      CacheArray[i-1].LogicalBlock:=Cur_Block
7050/    2B41 :                     ;      CacheArray[i-1].Head:=TempHead
7051/    2B41 :                     ;      CacheArray[i-1].Sector:=TempSector
7052/    2B41 :                     ;    Cache_Index:=0
7053/    2B41 :                     ;  End
7054/    2B41 :                     ;
7055/    2B41 :                     ;*****************************************************************
7056/    2B41 :
```

```
7057/    2B41 : 4C 14              Load_Cache      ld R4, #CacheLength
7058/    2B43 : 5C 00                              ld R5, #0               ; clear SeekNeeded, HeadChange booleans
7059/    2B45 : E6 58 02                           ld Data_Type, #User_Type
7060/    2B48 : 8C 16                              ld R8, #CacheStat /256
7061/    2B4A : 9C E8                              ld R9, #CacheStat #256
7062/    2B4C : AC 16                              ld R10, #CacheArray /256
7063/    2B4E : BC FC                              ld R11, #CacheArray #256
7064/    2B50 : D6 06 0D                           call Load_Logical       ; init logical block
7065/    2B53 :                    ;
7066/    2B53 : 2C 17                              ld R2, #TLBlock /256
7067/    2B55 : 3C 51                              ld R3, #TLBlock #256
7068/    2B57 : D6 2C 7B                           call St_Blk             ; store logical block into TLBlock
7069/    2B5A :                    ;
7070/    2B5A : D6 2C 83                           call Ld_C_Srch
7071/    2B5D : 70 FC                              push FLAGS              ; save result of spare table search
7072/    2B5F : 70 E0                              push R0
7073/    2B61 : 2C 17                              ld R2, #PBlock /256
7074/    2B63 : 3C 4E                              ld R3, #PBlock #256
7075/    2B65 : D6 2C 7B                           call St_Blk             ; store physical block into PBlock
7076/    2B68 : D6 2C 2F                           call Cache_Cnvrt
7077/    2B6B : 50 E0                              pop R0
7078/    2B6D : 50 FC                              pop FLAGS
7079/    2B6F : B0 E1                              clr R1                  ; SeekNeeded, HdChg:=false
7080/    2B71 : EB 1A                              jr NZ, Ld_Cache_In
7081/    2B73 : 8D 2B EB                           jp Ld_C_Enter
7082/    2B76 :                    ;
7083/    2B76 : 2C 17              Ld_Cache_Lp     ld R2, #PBlock /256
7084/    2B78 : 3C 4E                              ld R3, #PBlock #256
7085/    2B7A : D6 2C 5B                           call Inc_Blk            ; inc physical block
7086/    2B7D : 2C 17                              ld R2, #TLBlock /256
7087/    2B7F : 3C 51                              ld R3, #TLBlock #256
7088/    2B81 : D6 2C 5B                           call Inc_Blk            ; inc logical block
7089/    2B84 : D6 2C 83                           call Ld_C_Srch
7090/    2B87 : 6B 09                              jr Z, Cache_NoSpare
7091/    2B89 :                    ;
7092/    2B89 : 1C 80                              ld R1, #CachSeek
7093/    2B8B : 42 51                              or R5, R1               ; set SeekNeeded
7094/    2B8D : D6 2B F7           Ld_Cache_In     call Enter_Cache
7095/    2B90 : 8B 5E                              jr Ld_Cache_More
7096/    2B92 :                    ;
7097/    2B92 : 31 30              Cache_NoSpare   srp #Wrk_Sys2
7098/    2B94 :                            assume RP:Wrk_Sys2
7099/    2B94 : 98 1C                              ld R9, Wrk_Sys+12       ; nondestructive subtract
7100/    2B96 : A8 1D                              ld R10, Wrk_Sys+13
7101/    2B98 : B8 1E                              ld R11, Wrk_Sys+14
7102/    2B9A : 2C 17                              ld R2, #PBlock /256
7103/    2B9C : 3C 4E                              ld R3, #PBlock #256
7104/    2B9E : 0C 3C                              ld R0, #Wrk_Sys2+12
7105/    2BA0 : D6 2C 54                           call Gt_3_Ldei          ; get physical block
7106/    2BA3 : 22 EB                              sub R14, R11            ; If TLBlock <>PBlock
7107/    2BA5 : 32 DA                              sbc R13, R10
7108/    2BA7 : 32 C9                              sbc R12, R9
7109/    2BA9 : 42 CD                              or R12, R13
7110/    2BAB : 42 CE                              or R12, R14
7111/    2BAD : 70 FC                              push FLAGS
7112/    2BAF : 31 10                              srp #Wrk_Sys
7113/    2BB1 :                            assume RP:Wrk_Sys
7114/    2BB1 :                    ;
7115/    2BB1 : 2C 17                              ld R2, #PBlock /256
7116/    2BB3 : 3C 4E                              ld R3, #PBlock #256
7117/    2BB5 : D6 2C 7B                           call St_Blk
7118/    2BB8 : 50 FC                              pop FLAGS               ; check for spare block
7119/    2BBA : EB 20                              jr NZ, Ld_Cach_Seek
7120/    2BBC :                    ;
7121/    2BBC : D6 2C 4C                           call Gt_THS             ; load current seek address
7122/    2BBF : FE                                 inc R15                 ; bump the sector address
7123/    2BC0 : A6 EF 13                           cp R15, #NbrSctrs
7124/    2BC3 : 9B 07                              jr GE, Ld_Cach_HdChg
7125/    2BC5 :                    ;
7126/    2BC5 : D6 2C 3D                           call St_THS             ; save new seek address
7127/    2BC8 : B0 E1                              clr R1                  ; CacheSeek, CacheHdChg:=false
7128/    2BCA : 8B 1F                              jr Ld_C_Enter
7129/    2BCC :
7130/    2BCC : FC 00              Ld_Cach_HdChg   ld R15, #0              ; start at sector 0
7131/    2BCE : A6 EE 00                           cp R14, #0              ; check for head 0
7132/    2BD1 : EB 09                              jr NZ, Ld_Cach_Seek
7133/    2BD3 : EC 01                              ld R14, #1              ; otherwise go to head 1
7134/    2BD5 : D6 2C 3D                           call St_THS
7135/    2BD8 : 1C 40                              ld R1, #CachHdChg
7136/    2BDA : 8B 0F                              jr Ld_C_Enter
7137/    2BDC :                    ;
7138/    2BDC : EC 00              Ld_Cach_Seek    ld R14, #0              ; start at head 0
7139/    2BDE : FC 00                              ld R15, #0              ; and sector 0
7140/    2BE0 : 06 ED 01                           add R13, #1             ; bump the track count by 1
7141/    2BE3 : 16 EC 00                           adc R12, #0
7142/    2BE6 : D6 2C 3D                           call St_THS
7143/    2BE9 : 1C 80                              ld R1, #CachSeek
7144/    2BEB :                    ;
7145/    2BEB : 42 51              Ld_C_Enter      or R5, R1               ; set SeekNeeded or HeadChange
7146/    2BED : D6 2C 04                           call Ld_BlkStat
7147/    2BF0 : 4A 84              Ld_Cache_More   djnz R4, Ld_Cache_Lp
7148/    2BF2 : B0 5B              Ld_Cache_End    clr Cache_Index
7149/    2BF4 : 8D 04 F8                           jp Bank_Ret
7150/    2BF7 :                    ;
7151/    2BF7 :                    ;*****************************************************************
7152/    2BF7 :                    ;
7153/    2BF7 : 76 E0 10           Enter_Cache     tm R0, #Spare           ; check if block is a spare
7154/    2BFA : 6B 05                              jr Z, Ld_Blk_BB
7155/    2BFC : 46 E1 01                           or R1, #S_Block         ; set Spare Block status
7156/    2BFF : 8B 03                              jr Ld_BlkStat
7157/    2C01 : 46 E1 02           Ld_Blk_BB       or R1, #B_Block         ; otherwise it must be a Bad Block
7158/    2C04 :
7159/    2C04 : 42 15              Ld_BlkStat      or R1, R5               ; merge in global SeekNeeded or HeadChange
7160/    2C06 : 92 18                              lde @RR8, R1
7161/    2C08 : 2C 17                              ld R2, #TLBlock /256
7162/    2C0A : 3C 51                              ld R3, #TLBlock #256
7163/    2C0C : D6 2C 7F                           call Gt_Blk             ; get logical block
7164/    2C0F : 0C 1C                              ld R0, #Wrk_Sys+12      ; load into CacheArray.Logical
7165/    2C11 : 93 0A                              ldei @RR10, @R0
7166/    2C13 : 93 0A                              ldei @RR10, @R0
7167/    2C15 : 93 0A                              ldei @RR10, @R0
7168/    2C17 : A0 E8                              incw RR8                ; set up for the next go 'round
7169/    2C19 :                    :
7170/    2C19 : D6 2C 4C                           call Gt_THS
7171/    2C1C : 08 EF                              ld R0, R15
7172/    2C1E : D6 25 EF                           call ReMap_Sector
7173/    2C21 : F0 EE                              swap R14                ; merge Head with Sector
7174/    2C23 : CF                                 rcf
```

```
7175/    2C24 : 90 EE                            rl R14
7176/    2C26 : 90 EE                            rl R14
7177/    2C28 : 42 E0                            or R14, R0
7178/    2C2A : 92 EA                            lde @RR10, R14
7179/    2C2C : A0 EA                            incw RR10               ; point to next cache entry
7180/    2C2E : AF                               ret
7181/    2C2F :                          ;
7182/    2C2F : 2C 17                    Cache_Cnvrt     ld R2, #PBlock /256
7183/    2C31 : 3C 4E                            ld R3, #PBlock #256
7184/    2C33 : D6 2C 7F                         call Gt_Blk             ; get latest physical block
7185/    2C36 : D6 20 D4                         call Get_Cyl_H_S
7186/    2C39 : D6 2C 3D                         call St_THS             ; store latest seek address
7187/    2C3C : AF                               ret
7188/    2C3D :                          ;
7189/    2C3D : 2C 17                    St_THS          ld R2, #Cur_THS /256    ; store the current seek address
7190/    2C3F : 3C 54                            ld R3, #Cur_THS #256    ;  from R12:15
7191/    2C41 : 0C 1C                            ld R0, #Wrk_Sys+12
7192/    2C43 : 93 02                            ldei @RR2, @R0
7193/    2C45 : 93 02                    St_3_Ldei       ldei @RR2, @R0
7194/    2C47 : 93 02                            ldei @RR2, @R0
7195/    2C49 : 93 02                            ldei @RR2, @R0
7196/    2C4B : AF                               ret
7197/    2C4C :                          ;
7198/    2C4C : 2C 17                    Gt_THS          ld R2, #Cur_THS /256    ; load the current seek address
7199/    2C4E : 3C 54                            ld R3, #Cur_THS #256    ;  into R12:15
7200/    2C50 : 0C 1C                            ld R0, #Wrk_Sys+12
7201/    2C52 : 83 02                            ldei @R0, @RR2
7202/    2C54 : 83 02                    Gt_3_Ldei       ldei @R0, @RR2
7203/    2C56 : 83 02                            ldei @R0, @RR2
7204/    2C58 : 83 02                            ldei @R0, @RR2
7205/    2C5A : AF                               ret
7206/    2C5B :                          ;
7207/    2C5B : 0C 1C                    Inc_Blk         ld R0, #Wrk_Sys+12      ; get R12:14 from RAM
7208/    2C5D : 83 02                            ldei @R0, @RR2
7209/    2C5F : 83 02                            ldei @R0, @RR2
7210/    2C61 : 83 02                            ldei @R0, @RR2
7211/    2C63 : 06 EE 01                         add R14, #1             ; inc block number
7212/    2C66 : 16 ED 00                         adc R13, #0
7213/    2C69 : 16 EC 00                         adc R12, #0
7214/    2C6C : 26 E3 03                         sub R3, #3              ; restor RAM pointer to
7215/    2C6F : 36 E2 00                         sbc R2, #0              ;  original value
7216/    2C72 : 0C 1C                            ld R0, #Wrk_Sys+12      ; and write back into RAM
7217/    2C74 : 93 02                            ldei @RR2, @R0
7218/    2C76 : 93 02                            ldei @RR2, @R0
7219/    2C78 : 93 02                            ldei @RR2, @R0
7220/    2C7A : AF                               ret
7221/    2C7B :                          ;
7222/    2C7B : 0C 1C                    St_Blk          ld R0, #Wrk_Sys+12      ; copy R12:14 into
7223/    2C7D : 8B C6                            jr St_3_Ldei            ;  external memory @RR2
7224/    2C7F :                          ;
7225/    2C7F : 0C 1C                    Gt_Blk          ld R0, #Wrk_Sys+12      ; retrieve R12:14 from
7226/    2C81 : 8B D1                            jr Gt_3_Ldei            ;  external memory @RR2
7227/    2C83 :                          ;
7228/    2C83 :                          ;*********************************************************************
7229/    2C83 :                          ;  Fast search of spare table
7230/    2C83 :                          ;*********************************************************************
7231/    2C83 :                          ;
7232/    2C83 : 18 ED                    Ld_C_Srch       ld R1, R13              ; check if head ptr is NIL
7233/    2C85 : 08 EC                            ld R0, R12              ; but first form a headptr structure
7234/    2C87 : C0 E0                            rrc R0                  ; and index into HdPtr array
7235/    2C89 : C0 E1                            rrc R1
7236/    2C8B : CF                               rcf
7237/    2C8C : C0 E1                            rrc R1
7238/    2C8E : 2C 14                            ld R2, #SegPtrArray /256
7239/    2C90 : 3C C5                            ld R3, #SegPtrArray #256
7240/    2C92 : 02 31                            add R3, R1
7241/    2C94 : 16 E2 00                         adc R2, #0
7242/    2C97 : 82 02                            lde R0, @RR2            ; get HeadPtr and check for NIL
7243/    2C99 : 76 E0 80                         tm R0, #NIL
7244/    2C9C : 6B 1A                            jr Z, Ld_C_Long         ; do a real search if not NIL
7245/    2C9E : =>TRUE                   if W_10MB
7246/    2C9E : 08 ED                            ld R0, R13              ; save for possible rollover
7247/    2CA0 : 02 ED                            add R14, R13            ; Physical Block:= LBlock+LBlock div 256
7248/    2CA2 : 16 ED 00                         adc R13, #0
7249/    2CA5 : 16 EC 00                         adc R12, #0
7250/    2CA8 : [7245]                   endif
7251/    2CA8 : A2 0D                            cp R0, R13              ; check for rollover
7252/    2CAA : 6B 09                            jr Z, Ld_C_S_End
7253/    2CAC :                          ;
7254/    2CAC : 06 EE 01                         add R14, #1             ; otherwise bump Physical Block
7255/    2CAF : 16 ED 00                         adc R13, #0
7256/    2CB2 : 16 EC 00                         adc R12, #0
7257/    2CB5 :                          ;
7258/    2CB5 : B2 00                    Ld_C_S_End      xor R0, R0              ; return zero status
7259/    2CB7 : AF                       Ld_C_S_Ret      ret
7260/    2CB8 :                          ;
7261/    2CB8 : 2C 14                    Ld_C_Long       ld R2, #SrchSpTabl /256
7262/    2CBA : 3C 6E                            ld R3, #SrchSpTabl #256
7263/    2CBC : D6 04 AB                         call Bank_Call
7264/    2CBF : 8B F6                            jr Ld_C_S_Ret
7265/    2CC1 :
7266/    2CC1 :
7267/    2CC1 :
7268/    2CC1 :                          ;*********************************************************************
7269/    2CC1 :                          ; Module Progs.B1.Assem
7270/    2CC1 :                          ;
7271/    2CC1 :                          ; This module contains those routines that are 'high-level
7272/    2CC1 :                          ; programs
7273/    2CC1 :                          ;
7274/    2CC1 :                          ;       PROCEDURE Scan
7275/    2CC1 :                          ;       PROCEDURE Strt_FreeProcess
7276/    2CC1 :                          ;
7277/    2CC1 :                          ;*********************************************************************
7278/    2CC1 :
7279/    2CC1 :                          ;*********************************************************************
7280/    2CC1 :                          ;
7281/    2CC1 :                          ; Procedure: Scan
7282/    2CC1 :                          ;
7283/    2CC1 :                          ;  This procedure reads each logical block on the disk and
7284/    2CC1 :                          ;  spares them if they are bad.
7285/    2CC1 :                          ;
7286/    2CC1 :                          ; Inputs: none
7287/    2CC1 :                          ;
7288/    2CC1 :                          ; Outputs: none
7289/    2CC1 :                          ;
7290/    2CC1 :                          ; Algorithm:
7291/    2CC1 :                          ;
7292/    2CC1 :                          ;  Begin
```

```
7293/    2CC1 :                     ;    For i:=0 To MaxLogicalBlocks Do
7294/    2CC1 :                     ;      Seek_Type:=Access_Offset
7295/    2CC1 :                     ;      Data_Type:=User
7296/    2CC1 :                     ;      DiskStat.Wr_Op:=false
7297/    2CC1 :                     ;      If not(Srch_Cach(i) or Srch_Cach.SeekNeeded
7298/    2CC1 :                     ;        Then
7299/    2CC1 :                     ;          Seek(CnvrtLogical)
7300/    2CC1 :                     ;          Load_Cache
7301/    2CC1 :                     ;        Else
7302/    2CC1 :                     ;          If Srch_Cach.HdChg
7303/    2CC1 :                     ;            Then
7304/    2CC1 :                     ;              SelectHead(complement(Head))
7305/    2CC1 :                     ;              Load_Cache
7306/    2CC1 :                     ;            Else
7307/    2CC1 :                     ;              Sector:=Srch_Cach.Sector
7308/    2CC1 :                     ;     If not(Read_Common)
7309/    2CC1 :                     ;       Then Data_Ex_Handler(Read_Common.ErrorCode)
7310/    2CC1 :                     ;  End
7311/    2CC1 :                     ;
7312/    2CC1 :                     ;*****************************************************************
7313/    2CC1 :
7314/    2CC1 : D6 2E 13            D_Scan          call Scan_Clr
7315/    2CC4 : E6 0A 15                            ld Wrk_Io+10, #D_Scan_Response
7316/    2CC7 : D6 03 02                            call Ack_Read
7317/    2CCA :                     ;
7318/    2CCA : 76 25 01            Scan            tm SlfTst_Result, #No_SprTbl
7319/    2CCD : 6B 03                               jr Z, Scan_1
7320/    2CCF : D6 05 1F                            call Abort
7321/    2CD2 :                     ;
7322/    2CD2 : 0C 00               Scan_1          ld R0, #0
7323/    2CD4 : 1C 07                               ld R1, #7
7324/    2CD6 : 2C 19                               ld R2, #Wrk_Sys+9       ; init PBlock and Track-Head-Sector
7325/    2CD8 : F3 20               Scn_Init1       ld @R2, R0
7326/    2CDA : 2E                                  inc R2
7327/    2CDB : 1A FB                               djnz R1, Scn_Init1
7328/    2CDD : D6 2E 28                            call St_PBlock
7329/    2CE0 :                     ;
7330/    2CE0 : E6 57 80            Scan_Lp         ld Seek_Type, #Access
7331/    2CE3 : D6 26 05                            call Seek
7332/    2CE6 : 2C 12               Scan_Read       ld R2, #Read_Common /256
7333/    2CE8 : 3C F2                               ld R3, #Read_Common #256
7334/    2CEA : D6 04 AB                            call Bank_Call
7335/    2CED : EB 69                               jr NZ, Scan_More
7336/    2CEF : 70 E0               Scan_Except     push R0                 ; save error code
7337/    2CF1 : D6 22 15                            call Ext_Push
7338/    2CF4 : 1C 00               Scan_Cnvrt      ld R1, #0               ; convert to logical interleave
7339/    2CF6 : 2C 16                               ld R2, #Map_Table /256
7340/    2CF8 : 3C 81                               ld R3, #Map_Table #256
7341/    2CFA : 4C 13                               ld R4, #NbrSctrs
7342/    2CFC : 82 02               Scan_Map_Lp     lde R0, @RR2
7343/    2CFE : A4 55 E0                            cp R0, Sector
7344/    2D01 : 6B 08                               jr Z, Scan_Map_End
7345/    2D03 : A0 E2                               incw RR2
7346/    2D05 : 1E                                  inc R1
7347/    2D06 : 4A F4                               djnz R4, Scan_Map_Lp
7348/    2D08 : D6 05 1F                            call Abort
7349/    2D0B :                     ;
7350/    2D0B : D6 2E 35            Scan_Map_End    call Gt_PBlock
7351/    2D0E : 02 B1                               add R11, R1             ; add in the sector offset
7352/    2D10 : 16 EA 00                            adc R10, #0
7353/    2D13 : 16 E9 00                            adc R9, #0
7354/    2D16 : 0C 00                               ld R0, #0               ; check for logical block zero
7355/    2D18 : 42 0B                               or R0, R11
7356/    2D1A : 42 0A                               or R0, R10
7357/    2D1C : 42 09                               or R0, R9
7358/    2D1E : 6B 0C                               jr Z, Scan_Ld_L         ; block zero can not be a spare
7359/    2D20 :                     ; convert the physcal block number to a logical block number
7360/    2D20 : =>TRUE                      if W_10MB
7361/    2D20 : 42 BB                               or R11, R11             ; check for spareblock
7362/    2D22 : 6B 0E                               jr Z, Scn_Spare
7363/    2D24 : 22 BA                               sub R11, R10
7364/    2D26 : 36 EA 00                            sbc R10, #0
7365/    2D29 : 36 E9 00                            sbc R9, #0
7366/    2D2C : [7360]                      endif
7367/    2D2C : =>FALSE                     if W_20MB
7368/    2D2C :                                     ld R0, R10              ; check for spareblock
7369/    2D2C :                                     and R0, #01h
7370/    2D2C :                                     or R0, R11
7371/    2D2C :                                     jr Z, Scn_Spare
7372/    2D2C :                                     ld R1, R9
7373/    2D2C :                                     ld R2, R10
7374/    2D2C :                                     rcf
7375/    2D2C :                                     rrc R1
7376/    2D2C :                                     rrc R2
7377/    2D2C :                                     sub R11, R2
7378/    2D2C :                                     sbc R10, R1
7379/    2D2C :                                     sbc R9, #0
7380/    2D2C : [7367]                      endif
7381/    2D2C : =>FALSE                     if W_40MB
7382/    2D2C :                                     ld R0, R10              ; check for spareblock
7383/    2D2C :                                     and R0, #01h
7384/    2D2C :                                     or R0, R11
7385/    2D2C :                                     jr Z, Scn_Spare
7386/    2D2C :                                     ld R1, R9
7387/    2D2C :                                     ld R2, R10
7388/    2D2C :                                     ld R0, #2
7389/    2D2C :                     P_To_L          rcf
7390/    2D2C :                                     rrc R1
7391/    2D2C :                                     rrc R2
7392/    2D2C :                                     djnz R0, P_To_L
7393/    2D2C :                                     sub R11, R2
7394/    2D2C :                                     sbc R10, R1
7395/    2D2C :                                     sbc R9, #0
7396/    2D2C : [7381]                      endif
7397/    2D2C :                     ;
7398/    2D2C : D6 2E 1B            Scan_Ld_L       call Scan_LdLB
7399/    2D2F : 46 E0 01                            or R0, #1               ; set non-zero flag if not spare
7400/    2D32 : 6B 21               Scn_Spare       jr Z, Scan_M_Pop
7401/    2D34 : E6 58 02                            ld Data_Type, #User_Type
7402/    2D37 : D6 2B 41                            call Load_Cache
7403/    2D3A : D6 06 0D                            call Load_Logical
7404/    2D3D : 2C 10                               ld R2, #SC_Vector /256
7405/    2D3F : 3C 88                               ld R3, #SC_Vector #256
7406/    2D41 : D6 04 AB                            call Bank_Call
7407/    2D44 : 50 E0                               pop R0                  ; retrieve error code
7408/    2D46 : 76 5A 01                            tm BlkStat, #S_Block    ; check for Spare block
7409/    2D49 : EB 0A                               jr NZ, Scan_M_Pop
7410/    2D4B :                     ;
```

```
7411/    2D4B : 2C 16              Scan_Rd_Err     ld R2, #Data_Ex_Handler /256
7412/    2D4D : 3C B1                              ld R3, #Data_Ex_Handler #256
7413/    2D4F : D6 04 AB                           call Bank_Call
7414/    2D52 : D6 22 8B                           call ZeroHeader
7415/    2D55 :                    ;
7416/    2D55 : D6 22 36           Scan_M_Pop      call Ext_Pop
7417/    2D58 :                    ;
7418/    2D58 : 20 55              Scan_More       inc Sector              ; read next sector
7419/    2D5A : A6 55 13                           cp Sector, #NbrSctrs    ; check for overflow
7420/    2D5D : 1B 87                              jr LT, Scan_Read
7421/    2D5F : D6 2E 35                           call Gt_PBlock
7422/    2D62 : 06 EB 13                           add R11, #NbrSctrs      ; go to next set of blocks
7423/    2D65 : 16 EA 00                           adc R10, #0
7424/    2D68 : 16 E9 00                           adc R9, #0
7425/    2D6B : D6 2E 28                           call St_PBlock
7426/    2D6E : E6 55 00                           ld Sector, #0           ; otherwise start at sector 0
7427/    2D71 : 20 54                              inc Head                ; and go to next head
7428/    2D73 : A6 54 02                           cp Head, #NbrHds
7429/    2D76 : 9B 08                              jr GE, Scan_IncTrks
7430/    2D78 : E8 54                              ld R14, Head
7431/    2D7A : D6 27 17                           call SelectHead
7432/    2D7D : 8D 2C E6                           jp Scan_Read
7433/    2D80 :                    ;
7434/    2D80 : EC 00              Scan_IncTrks    ld R14, #0
7435/    2D82 : FC 00                              ld R15, #0
7436/    2D84 : D8 53                              ld R13, Cylinder+1
7437/    2D86 : C8 52                              ld R12, Cylinder
7438/    2D88 : 06 ED 01                           add R13, #1             ; otherwise seek to the next track
7439/    2D8B : 16 EC 00                           adc R12, #0
7440/    2D8E : 0C 02                              ld R0, #(NbrTracks-1) /256
7441/    2D90 : 1C 01                              ld R1, #(NbrTracks-1) #256
7442/    2D92 : B2 1D                              xor R1, R13             ; check for last track
7443/    2D94 : B2 0C                              xor R0, R12
7444/    2D96 : 42 01                              or R0, R1
7445/    2D98 : ED 2C E0                           jp NZ, Scan_Lp
7446/    2D9B :                    ;
7447/    2D9B : 4C 00              Scan_End        ld R4, #0               ; start searching for bad blocks
7448/    2D9D : 2C 14              Scn_BadLp:      ld R2, #SegPtrArray /256
7449/    2D9F : 3C C5                              ld R3, #SegPtrArray #256
7450/    2DA1 : 02 34                              add R3, R4              ; get index to HeadPtr
7451/    2DA3 : 16 E2 00           Scn_Bd3         adc R2, #0
7452/    2DA6 : 82 02                              lde R0, @RR2            ; get HeadPtr
7453/    2DA8 : 76 E0 80                           tm R0, #Nil             ; check for any elements here
7454/    2DAB : EB 57                              jr NZ, Scn_BdNext
7455/    2DAD :                    ;
7456/    2DAD : 56 E0 7F           Scn_Bd1         and R0, #0FFh-Nil       ; mask off NIL bit
7457/    2DB0 : 2C 17                              ld R2, #Get_Ptr /256
7458/    2DB2 : 3C F8                              ld R3, #Get_Ptr #256
7459/    2DB4 : D6 04 AB                           call Bank_Call
7460/    2DB7 : 82 02                              lde R0, @RR2            ; get status of element
7461/    2DB9 : 76 E0 02                           tm R0, #User_Type       ; check only user data
7462/    2DBC : 6B 3C                              jr Z, Scn_Bd2
7463/    2DBE : 76 E0 10                           tm R0, #Spare           ; look at only badblocks
7464/    2DC1 : EB 37                              jr NZ, Scn_Bd2
7465/    2DC3 : D6 22 15                           call Ext_Push           ; get logical block
7466/    2DC6 : 9C 00                              ld R9, #0
7467/    2DC8 : 90 E4                              rl R4
7468/    2DCA : 10 E4                              rlc R4
7469/    2DCC : 10 E9                              rlc R9
7470/    2DCE : A0 E2                              incw RR2
7471/    2DD0 : 82 A2                              lde R10, @RR2
7472/    2DD2 : 42 A4                              or R10, R4
7473/    2DD4 : A0 E2                              incw RR2
7474/    2DD6 : 82 B2                              lde R11, @RR2
7475/    2DD8 : D6 2E 1B                           call Scan_LdLB
7476/    2DDB :                    ;
7477/    2DDB : D6 06 0D                           call Load_Logical
7478/    2DDE : 2C 16                              ld R2, #CnvrtLogical /256
7479/    2DE0 : 3C 3A                              ld R3, #CnvrtLogical #256
7480/    2DE2 : D6 04 AB                           call Bank_Call
7481/    2DE5 : D6 26 05                           call Seek
7482/    2DE8 : 0C 84                              ld R0, #Error+Ex_BadBlock
7483/    2DEA : E6 58 02                           ld Data_Type, #User_Type
7484/    2DED : E6 5A 02                           ld BlkStat, #B_Block
7485/    2DF0 : 2C 16                              ld R2, #Data_Ex_Handler /256
7486/    2DF2 : 3C B1                              ld R3, #Data_Ex_Handler #256
7487/    2DF4 : D6 04 AB                           call Bank_Call
7488/    2DF7 : D6 22 36                           call Ext_Pop
7489/    2DFA :
7490/    2DFA : 76 E0 80           Scn_Bd2         tm R0, #Nil             ; check for end of chain
7491/    2DFD : EB 05                              jr NZ, Scn_BdNext
7492/    2DFF : 06 E3 03                           add R3, #3              ; get ptr to next
7493/    2E02 : 8B 9F                              jr Scn_Bd3
7494/    2E04 :                    ;
7495/    2E04 : 4E                 Scn_BdNext      inc R4                  ; go to next HeadPtr
7496/    2E05 : =>TRUE                     if W_10MB
7497/    2E05 : A6 E4 20                           cp R4, #20h             ; only 32 HeadPtrs for 10MB
7498/    2E08 : [7496]                     endif
7499/    2E08 : =>FALSE                    if W_20MB
7500/    2E08 :                                    cp R4, #40h             ; 64 HeadPtrs for 20MB
7501/    2E08 : [7499]                     endif
7502/    2E08 : =>FALSE                    if W_40MB
7503/    2E08 :                                    cp R4, #80h             ; 128 HeadPtrs for 40MB
7504/    2E08 : [7502]                     endif
7505/    2E08 :
7506/    2E08 : EB 93                              jr NZ, Scn_BadLp
7507/    2E0A : D6 29 88                           call Park_Heads
7508/    2E0D : D6 2E 13                           call Scan_Clr
7509/    2E10 : 8D 04 F8                           jp Bank_Ret
7510/    2E13 :                    ;
7511/    2E13 :                    ;***************************
7512/    2E13 :                    ;
7513/    2E13 : 2C 1E              Scan_Clr        ld R2, #ClrNormStat /256
7514/    2E15 : 3C 5F                              ld R3, #ClrNormStat #256
7515/    2E17 : D6 04 AB                           call Bank_Call
7516/    2E1A : AF                                 ret
7517/    2E1B :                    ;
7518/    2E1B : 2C 14              Scan_LdLB       ld R2, #LogicalBlock /256
7519/    2E1D : 3C A1                              ld R3, #LogicalBlock #256
7520/    2E1F : 1C 04                              ld R1, #4
7521/    2E21 : 0C 19                              ld R0, #Wrk_Sys+9
7522/    2E23 : 93 02              Scan_LB_Lp      ldei @RR2, @R0
7523/    2E25 : 1A FC                              djnz R1, Scan_LB_Lp
7524/    2E27 : AF                                 ret
7525/    2E28 :                    ;
7526/    2E28 : 2C 17              St_PBlock       ld R2, #PBlk /256
7527/    2E2A : 3C 58                              ld R3, #PBlk #256
7528/    2E2C : 0C 19                              ld R0, #Wrk_Sys+9
```

```
7529/    2E2E : 93 02                        ldei @RR2, @R0
7530/    2E30 : 93 02                        ldei @RR2, @R0
7531/    2E32 : 93 02                        ldei @RR2, @R0
7532/    2E34 : AF                           ret
7533/    2E35 :                  ;
7534/    2E35 : 2C 17            Gt_PBlock   ld R2, #PBlk /256
7535/    2E37 : 3C 58                        ld R3, #PBlk #256
7536/    2E39 : 0C 19                        ld R0, #Wrk_Sys+9
7537/    2E3B : 83 02                        ldei @R0, @RR2
7538/    2E3D : 83 02                        ldei @R0, @RR2
7539/    2E3F : 83 02                        ldei @R0, @RR2
7540/    2E41 : AF                           ret
7541/    2E42 :
7542/    2E42 :
7543/    2E42 :                  ;*****************************************************************
7544/    2E42 :                  ;
7545/    2E42 :                  ; Procedure: Strt_FreeProcess
7546/    2E42 :                  ;
7547/    2E42 :                  ;  This procedure is used by the controller to start any process
7548/    2E42 :                  ;  (along the lines of internal bookkeeping) that it choses
7549/    2E42 :                  ;
7550/    2E42 :                  ; Inputs: none
7551/    2E42 :                  ;
7552/    2E42 :                  ; Outputs: none
7553/    2E42 :                  ;
7554/    2E42 :                  ; Algorithm:
7555/    2E42 :                  ;
7556/    2E42 :                  ;  Begin
7557/    2E42 :                  ;    If SeekCount>=2048
7558/    2E42 :                  ;      Then ArmSweep
7559/    2E42 :                  ;    Free_SlfTst:=0
7560/    2E42 :                  ;    While not(CMD) Do
7561/    2E42 :                  ;      Bsy_Wait for 2 seconds
7562/    2E42 :                  ;      If not(Parked) Then Park_Heads
7563/    2E42 :                  ;      Case Free_Slftst Of
7564/    2E42 :                  ;        0: Test Eprom
7565/    2E42 :                  ;        1: Test Sector Count
7566/    2E42 :                  ;        2: Test Motor Speed
7567/    2E42 :                  ;        3: Test Read/Write; Free_SlfTst:=0
7568/    2E42 :                  ;    Take care of Host
7569/    2E42 :                  ;  End
7570/    2E42 :                  ;
7571/    2E42 :                  ;*****************************************************************
7572/    2E42 :
7573/    2E42 : 8F               Strt_FreeProcess    di
7574/    2E43 : 31 10                        srp #Wrk_Sys            ; get into a reasonable state
7575/    2E45 :                          assume RP:Wrk_Sys
7576/    2E45 : B0 FE                        clr SPH
7577/    2E47 : E6 FF 80                     ld SPL, #Stack_Top
7578/    2E4A : 56 02 F7                     and P2, #0FFh-Bsy       ; clear BSY
7579/    2E4D : 76 2A F8                     tm SeekCount, #0F8h     ; do arm sweep every 2k seeks or so
7580/    2E50 : 6B 20                        jr Z, Chk_Park
7581/    2E52 :                  ;
7582/    2E52 : B0 2A                        clr SeekCount
7583/    2E54 : B0 2B                        clr SeekCount+1
7584/    2E56 : CC 01                        ld R12, #Init_HiCyl     ; seek to data recal location
7585/    2E58 : DC F9                        ld R13, #Init_LoCyl
7586/    2E5A : EC 00                        ld R14, #0              ; head zero
7587/    2E5C : FC 00                        ld R15, #0              ; sector zero
7588/    2E5E : D6 05 D7                     call New_Seek
7589/    2E61 : CC 00                        ld R12, #0
7590/    2E63 : DC 40                        ld R13, #64
7591/    2E65 : EC 00                        ld R14, #0
7592/    2E67 : FC 00                        ld R15, #0
7593/    2E69 : D6 05 D7                     call New_Seek
7594/    2E6C : 56 56 7F                     and DiskStat, #0FFh-On_Track
7595/    2E6F : D6 2A 0E                     call Set_SeekNeeded     ; invalidate cache contents
7596/    2E72 :                  ;
7597/    2E72 : B0 23            Chk_Park    clr Free_SlfTst         ; get ready to do some self tests
7598/    2E74 : 2C 00                        ld R2, #200 /256        ; wait for two seconds
7599/    2E76 : 3C C8                        ld R3, #200 #256
7600/    2E78 : 8B 04                        jr Chk_Pk_Int
7601/    2E7A :                  ;
7602/    2E7A : 2C 03            Chk_Pk_Jp   ld R2, #800 /256        ; wait for eight seconds
7603/    2E7C : 3C 20                        ld R3, #800 #256
7604/    2E7E : D6 04 21         Chk_Pk_Int  call Chk_Park1          ; do busy wait while monitoring CMD
7605/    2E81 :                  ;
7606/    2E81 : 76 56 10         Chk_Park3   tm DiskStat, #Parked
7607/    2E84 : EB 05                        jr NZ, FreeP_NoPark
7608/    2E86 : D6 29 88                     call Park_Heads
7609/    2E89 : 8B EF                        jr Chk_Pk_Jp
7610/    2E8B :                  ;
7611/    2E8B : 76 23 01         FreeP_NoPark    tm Free_SlfTst, #1  ; check for even value
7612/    2E8E : 6B 04                        jr Z, FreeP_SlfTst      ; do tests only every other time
7613/    2E90 : 20 23                        inc Free_SlfTst         ; otherwise make even for the next time
7614/    2E92 : 8B E6                        jr Chk_Pk_Jp
7615/    2E94 :                  ;
7616/    2E94 : 2C 2E            FreeP_SlfTst    ld R2, #SlfTst_Table /256
7617/    2E96 : 3C A8                        ld R3, #SlfTst_Table #256
7618/    2E98 : 04 23 E3                     add R3, Free_SlfTst
7619/    2E9B : 16 E2 00                     adc R2, #0
7620/    2E9E : 20 23                        inc Free_SlfTst         ; make value odd
7621/    2EA0 : C2 E2                        ldc R14, @RR2
7622/    2EA2 : A0 E2                        incw RR2
7623/    2EA4 : C2 F2                        ldc R15, @RR2
7624/    2EA6 : 30 EE                        jp @RR14
7625/    2EA8 :                  ;
7626/    2EA8 : 2E BA            SlfTst_Table    DW Free_Ram
7627/    2EAA : 2E 7A                        DW Chk_Pk_Jp            ; nop
7628/    2EAC : 2E C8                        DW Free_Eprom
7629/    2EAE : 2E 7A                        DW Chk_Pk_Jp
7630/    2EB0 : 2E E2                        DW Free_MtrSpd
7631/    2EB2 : 2E 7A                        DW Chk_Pk_Jp
7632/    2EB4 : 2E D4                        DW Free_SctrCnt
7633/    2EB6 : 2E 7A                        DW Chk_Pk_Jp
7634/    2EB8 : 2E F1                        DW Free_Rw
7635/    2EBA :                  ;
7636/    2EBA : 2C 24            Free_Ram    ld R2, #Chk_SprChk /256 ; verify spare table checksum
7637/    2EBC : 3C 05                        ld R3, #Chk_SprChk #256
7638/    2EBE : D6 04 AB                     call Bank_Call
7639/    2EC1 : EB B7                        jr NZ, Chk_Pk_Jp
7640/    2EC3 :                  ;
7641/    2EC3 : 46 25 80                     or SlfTst_Result, #Ram_Fail
7642/    2EC6 : EB B2                        jr NZ, Chk_Pk_Jp
7643/    2EC8 :                  ;
7644/    2EC8 : 0C 02            Free_Eprom  ld R0, #Eprom2
7645/    2ECA : D6 01 89                     call EpromTest
7646/    2ECD : 6B AB                        jr Z, Chk_Pk_Jp
```

```
7647/    2ECF :                         ;
7648/    2ECF : 46 25 40                        or SlfTst_Result, #Eprom_Fail
7649/    2ED2 : 8B A6                           jr Chk_Pk_Jp
7650/    2ED4 :                         ;
7651/    2ED4 : 2C 2A                   Free_SctrCnt    ld R2, #SctrCount /256
7652/    2ED6 : 3C C8                           ld R3, #SctrCount #256
7653/    2ED8 : D6 04 AB                        call Bank_Call
7654/    2EDB : 6B 9D                           jr Z, Chk_Pk_Jp
7655/    2EDD :                         ;
7656/    2EDD : 46 25 08                        or SlfTst_Result, #Sector_Cnt
7657/    2EE0 : 8B 98                           jr Chk_Pk_Jp
7658/    2EE2 :                         ;
7659/    2EE2 : 2C 2A                   Free_MtrSpd     ld R2, #MtrSpd /256
7660/    2EE4 : 3C 6D                           ld R3, #MtrSpd #256
7661/    2EE6 : D6 04 AB                        call Bank_Call
7662/    2EE9 : 6B 8F                           jr Z, Chk_Pk_Jp
7663/    2EEB :                         ;
7664/    2EEB : 46 25 20                        or SlfTst_Result, #Disk_Speed
7665/    2EEE : 8D 2E 7A                        jp Chk_Pk_Jp
7666/    2EF1 :                         ;
7667/    2EF1 : 46 25 02                Free_RW         or SlfTst_Result, #Rw_Fail      ; assume failure
7668/    2EF4 : 2C 2A                           ld R2, #RwTest1 /256
7669/    2EF6 : 3C F5                           ld R3, #RwTest1 #256
7670/    2EF8 : D6 04 AB                        call Bank_Call
7671/    2EFB : 6D 2E 7A                        jp Z, Chk_Pk_Jp
7672/    2EFE :                         ;
7673/    2EFE : 56 25 FD                        and SlfTst_Result, #0FFh-Rw_Fail
7674/    2F01 : B0 23                           clr Free_SlfTst
7675/    2F03 : 8D 2E 42                        jp Strt_FreeProcess     ; check for arm sweep
7676/    2F06 :
7677/    2F06 :
7678/    2F06 :
7679/    2F06 :                         ;*****************************************************************
7680/    2F06 :                         ; Module ECC.B1.Assem   {Error Check and Correction}
7681/    2F06 :                         ;
7682/    2F06 :                         ; This module contains all the relevant files pertaining
7683/    2F06 :                         ; to the ECC method and algorithm used on Widget
7684/    2F06 :                         ;
7685/    2F06 :                         ;       FUNCTION Ecc : BOOLEAN
7686/    2F06 :                         ;       PROCEDURE ShiftAndXor(VAR R1,R2,R3,R4,R5,R6 : BYTE {R1:6})
7687/    2F06 :                         ;       PROCEDURE TestMod8(J : WORD {RR8})
7688/    2F06 :                         ;       PROCEDURE Test0(J : WORD {RR8})
7689/    2F06 :                         ;
7690/    2F06 :                         ;*****************************************************************
7691/    2F06 :
7692/    2F06 :                         ;*****************************************************************
7693/    2F06 :                         ;
7694/    2F06 :                         ; Function: Ecc
7695/    2F06 :                         ;
7696/    2F06 :                         ;  This function is responsible for 1) checking if the data in the
7697/    2F06 :                         ;  ReadBuffer is correctable and 2) correcting that data if it is
7698/    2F06 :                         ;  correctable.
7699/    2F06 :                         ;
7700/    2F06 :                         ;  The method used was prepared by:
7701/    2F06 :                         ;    Neil Glover
7702/    2F06 :                         ;    Daza Systems Technology Corp.
7703/    2F06 :                         ;    1801 Aspen St.
7704/    2F06 :                         ;    Broomfield, Co. 80020
7705/    2F06 :                         ;    (303) 466-5228
7706/    2F06 :                         ;
7707/    2F06 :                         ; Inputs:  none
7708/    2F06 :                         ;
7709/    2F06 :                         ; Outputs: Ecc: BOOLEAN {zero flag is set if not correctable}
7710/    2F06 :                         ;
7711/    2F06 :                         ; K1 = BitLength(DataField)+BitLength(CrcField)+BitLength(EccField)-41
7712/    2F06 :                         ;    = (1+532)*8 + 2*8 + 6*8 -41
7713/    2F06 :                         ;    = 4287
7714/    2F06 :                         ;
7715/    2F06 :                         ; Correction Span = 12 bits
7716/    2F06 :                         ;
7717/    2F06 :                         ;  R2Mask = $00
7718/    2F06 :                         ;  R3Mask = $0F
7719/    2F06 :                         ;
7720/    2F06 :                         ; Syndrome Bytes begin at ReadArray.RBuf1Ecc
7721/    2F06 :                         ;
7722/    2F06 :                         ; Local Variables: R1:          BYTE {R1}
7723/    2F06 :                         ;                  R2:          BYTE {R2}
7724/    2F06 :                         ;                  R3:          BYTE {R3}
7725/    2F06 :                         ;                  R4:          BYTE {R4}
7726/    2F06 :                         ;                  R5:          BYTE {R5}
7727/    2F06 :                         ;                  R6:          BYTE {R6}
7728/    2F06 :                         ;                  Correctable: BOOLEAN {R7/bit 7}
7729/    2F06 :                         ;                  Aligned:     BOOLEAN {R7/bit 6}
7730/    2F06 :                         ;                  Done:        BOOLEAN {R7/bit 5}
7731/    2F06 :                         ;                  J:           WORD {RR8}
7732/    2F06 :                         ;
7733/    2F06 :                         ; Algorithm:
7734/    2F06 :                         ;
7735/    2F06 :                         ;  Begin
7736/    2F06 :                         ;    R1:=SynfromeByte[1] {most significant byte}
7737/    2F06 :                         ;    R2:=SynfromeByte[2]
7738/    2F06 :                         ;    R3:=SynfromeByte[3]
7739/    2F06 :                         ;    R4:=SynfromeByte[4]
7740/    2F06 :                         ;    R5:=SynfromeByte[5]
7741/    2F06 :                         ;    R6:=SynfromeByte[6]
7742/    2F06 :                         ;    If (R1=R2=R3=R4=R5=R6=0)
7743/    2F06 :                         ;      Then Ecc:=false
7744/    2F06 :                         ;      Else
7745/    2F06 :                         ;        J:=K1
7746/    2F06 :                         ;        Aligned:=false
7747/    2F06 :                         ;        Done:=false
7748/    2F06 :                         ;        Correctable:=false
7749/    2F06 :                         ;        While R1=0 Do
7750/    2F06 :                         ;          ShiftRegsLeft 1 Byte {left-hand justify syndrome}
7751/    2F06 :                         ;          J:=J+8
7752/    2F06 :                         ;        While not(Done) Or not(Aligned) Do
7753/    2F06 :                         ;          ShiftAndXor
7754/    2F06 :                         ;          If R1=0
7755/    2F06 :                         ;            Then
7756/    2F06 :                         ;              If R4=R5=R6=0 And R2*R2Mask=0 And R3*R3Mask
7757/    2F06 :                         ;                Then
7758/    2F06 :                         ;                  Aligned:=true
7759/    2F06 :                         ;                  TestMod8
7760/    2F06 :                         ;            If not(Aligned) Then Test0
7761/    2F06 :                         ;            If not(Done) Then J:=J-1
7762/    2F06 :                         ;        While not(Done) Do
7763/    2F06 :                         ;          ShiftAndXor
7764/    2F06 :                         ;          If R=0
```

```
7765/    2F06 :                    ;            Then TestMod8
7766/    2F06 :                    ;            Else Test0
7767/    2F06 :                    ;         If not(Done) Then J:=J-1
7768/    2F06 :                    ;       If Correctable
7769/    2F06 :                    ;         Then
7770/    2F06 :                    ;           J:=J div 8
7771/    2F06 :                    ;           Buffer1[J]:=Buffer1[J] Xor R2
7772/    2F06 :                    ;           Buffer1[J+1]:=Buffer1[J+1] Xor R3
7773/    2F06 :                    ;           Buffer1[J+2]:=Buffer1[J+2] Xor R4
7774/    2F06 :                    ;           BlockMove(Buffer2, RBuffer
7775/    2F06 :                    ;       Ecc:=Correctable
7776/    2F06 :                    ;  End
7777/    2F06 :                    ;
7778/    2F06 :                    ;*********************************************************************
7779/    2F06 :
7780/    2F06 : =10BFH             K1              EQU 4287
7781/    2F06 : =0H                R2Mask          EQU 0
7782/    2F06 : =FH                R3Mask          EQU 00Fh
7783/    2F06 : =80H               Ecc_Correctable EQU 080h
7784/    2F06 : =40H               Ecc_Aligned     EQU 040h
7785/    2F06 : =20H               Ecc_Done        EQU 020h
7786/    2F06 :
7787/    2F06 : B0 E7              ECC             clr R7                  ; clear booleans
7788/    2F08 : 8C 10                              ld R8, #K1 /256
7789/    2F0A : 9C BF                              ld R9, #K1 #256
7790/    2F0C : 0C 00                              ld R0, #0               ; get ready to check for all zero syndrome
7791/    2F0E : BC 06                              ld R11, #6              ; load six bytes, R1..R6:=Syndrome Bytes
7792/    2F10 : CC 12                              ld R12, #RBuf1Ecc /256
7793/    2F12 : DC 2F                              ld R13, #RBuf1Ecc #256
7794/    2F14 : AC 11                              ld R10, #Wrk_Sys+1      ; load syndrome bytes into registers
7795/    2F16 :                    ;
7796/    2F16 : 83 AC              Ecc_Ld_Lp       ldei @R10, @RR12
7797/    2F18 : 43 0A                              or R0, @R10
7798/    2F1A : BA FA                              djnz R11, Ecc_Ld_Lp
7799/    2F1C : 6D 2F AB                           jp Z, Ecc_End
7800/    2F1F :                    ;
7801/    2F1F : 42 11              Ecc_LHJ_While   or R1, R1               ; While R1=0 Do
7802/    2F21 : EB 14                              jr NZ, Ecc_Align
7803/    2F23 : 18 E2                              ld R1, R2               ; shift left 1 byte
7804/    2F25 : 28 E3                              ld R2, R3
7805/    2F27 : 38 E4                              ld R3, R4
7806/    2F29 : 48 E5                              ld R4, R5
7807/    2F2B : 58 E6                              ld R5, R6
7808/    2F2D : B0 E6                              clr R6
7809/    2F2F : 06 E9 08                           add R9, #8              ; J:=J+8
7810/    2F32 : 16 E8 00                           adc R8, #0
7811/    2F35 : 8B E8                              jr Ecc_LHJ_While
7812/    2F37 :                    ;
7813/    2F37 : D6 2F AE           Ecc_Align       call ShiftAndXor
7814/    2F3A : EB 15                              jr NZ, Ecc_Al_1
7815/    2F3C : 08 E4                              ld R0, R4               ; If (R4=R5=R6=0)
7816/    2F3E : 42 05                              or R0, R5
7817/    2F40 : 42 06                              or R0, R6
7818/    2F42 : F8 E3                              ld R15, R3              ; And (R3*R3Mask=0)
7819/    2F44 : 56 EF 0F                           and R15, #R3Mask
7820/    2F47 : 42 0F                              or R0, R15
7821/    2F49 : EB 06                              jr NZ, Ecc_Al_1
7822/    2F4B :                    ;
7823/    2F4B : 46 E7 40                           or R7, #Ecc_Aligned
7824/    2F4E : D6 2F CF                           call TestMod8
7825/    2F51 :                    ;
7826/    2F51 : 76 E7 40           Ecc_Al_1        tm R7, #Ecc_Aligned
7827/    2F54 : EB 03                              jr NZ, Ecc_Al_2
7828/    2F56 :                    ;
7829/    2F56 : D6 2F DA                           call Test0
7830/    2F59 :                    ;
7831/    2F59 : 76 E7 20           Ecc_Al_2        tm R7, #20h
7832/    2F5C : EB 1D                              jr NZ, Ecc_Crct
7833/    2F5E :                    ;
7834/    2F5E : 80 E8                              decw RR8                ; J:=J-1
7835/    2F60 : 76 E7 60                           tm R7, #Ecc_Done+Ecc_Aligned
7836/    2F63 : 6B D2                              jr Z, Ecc_Align
7837/    2F65 :                    ;
7838/    2F65 : D6 2F AE           Ecc_Shift       call ShiftAndXor
7839/    2F68 : EB 05                              jr NZ, Ecc_Shft_Else
7840/    2F6A : D6 2F CF                           call TestMod8
7841/    2F6D : 8B 03                              jr Ecc_Shft_2
7842/    2F6F :                    ;
7843/    2F6F : D6 2F DA           Ecc_Shft_Else   call Test0
7844/    2F72 : 76 E7 20           Ecc_Shft_2      tm R7, #Ecc_Done
7845/    2F75 : EB 04                              jr NZ, Ecc_Crct
7846/    2F77 : 80 E8                              decw RR8                ; J:=J-1
7847/    2F79 : 8B EA                              jr Ecc_Shift
7848/    2F7B :                    ;
7849/    2F7B : 76 E7 80           Ecc_Crct        tm R7, #Ecc_Correctable
7850/    2F7E : 6B 2B                              jr Z, Ecc_End
7851/    2F80 :                    ;
7852/    2F80 : AC 03                              ld R10, #3              ; J:=J div 8
7853/    2F82 : C0 E8              Ecc_Div8        rrc R8
7854/    2F84 : C0 E9                              rrc R9
7855/    2F86 : AA FA                              djnz R10, Ecc_Div8
7856/    2F88 : 56 E8 1F                           and R8, #1Fh            ; mask off unwanted carries
7857/    2F8B : CC 10                              ld R12, #RDummy /256
7858/    2F8D : DC 18                              ld R13, #RDummy #256
7859/    2F8F : 02 D9                              add R13, R9
7860/    2F91 : 12 C8                              adc R12, R8
7861/    2F93 : BC 12                              ld R11, #Wrk_Sys+2      ; start with R2
7862/    2F95 : AC 03                              ld R10, #3              ; correct 3 bytes
7863/    2F97 :                    ;
7864/    2F97 : 82 1C              Ecc_Crct_Lp     lde R1, @RR12
7865/    2F99 : B3 1B                              xor R1, @R11
7866/    2F9B : F3 B1                              ld @R11, R1
7867/    2F9D : 93 BC                              ldei @RR12, @R11
7868/    2F9F : AA F6                              djnz R10, Ecc_Crct_Lp
7869/    2FA1 :                    ;
7870/    2FA1 : 2C 20                              ld R2, #RBuf_To_Buf2 /256
7871/    2FA3 : 3C 99                              ld R3, #RBuf_To_Buf2 #256
7872/    2FA5 : D6 04 AB                           call Bank_Call
7873/    2FA8 : 76 E7 80                           tm R7, #Ecc_Correctable ; set correctable flag
7874/    2FAB :                    ;
7875/    2FAB : 8D 04 F8           Ecc_End         jp Bank_Ret
7876/    2FAE :
7877/    2FAE :
7878/    2FAE :                    ;*********************************************************************
7879/    2FAE :                    ;
7880/    2FAE :                    ; Procedure: ShiftAndXor
7881/    2FAE :                    ;
7882/    2FAE :                    ;  This procedure is used to shift the current syndrome bytes
```

```
7883/    2FAE :                     ;  (assumed to be located in R1:6) to the right 1 bit and then
7884/    2FAE :                     ;  Xor the syndromes with the reciprocal polynominal if needed.
7885/    2FAE :                     ;
7886/    2FAE :                     ; Inputs: R1: BYTE {R1}
7887/    2FAE :                     ;         R2: BYTE {R2}
7888/    2FAE :                     ;         R3: BYTE {R3}
7889/    2FAE :                     ;         R4: BYTE {R4}
7890/    2FAE :                     ;         R5: BYTE {R5}
7891/    2FAE :                     ;         R6: BYTE {R6}
7892/    2FAE :                     ;
7893/    2FAE :                     ; Outputs: R1: BYTE {R1}
7894/    2FAE :                     ;          R2: BYTE {R2}
7895/    2FAE :                     ;          R3: BYTE {R3}
7896/    2FAE :                     ;          R4: BYTE {R4}
7897/    2FAE :                     ;          R5: BYTE {R5}
7898/    2FAE :                     ;          R6: BYTE {R6}
7899/    2FAE :                     ;
7900/    2FAE :                     ; Algorithm:
7901/    2FAE :                     ;
7902/    2FAE :                     ;  Begin
7903/    2FAE :                     ;    Shift the syndromes right 1 bit with carry
7904/    2FAE :                     ;    If the LSB of R6 was a 1 {if carry}
7905/    2FAE :                     ;      Then
7906/    2FAE :                     ;        R1:=R1 Xor 140
7907/    2FAE :                     ;        R2:=R3 Xor 12
7908/    2FAE :                     ;        R3:=R3 Xor 10
7909/    2FAE :                     ;        R4:=R4 Xor 40
7910/    2FAE :                     ;        R5:=R5 Xor 24
7911/    2FAE :                     ;        R6:=R6 Xor 8
7912/    2FAE :                     ;  End
7913/    2FAE :                     ;
7914/    2FAE :                     ;*****************************************************************
7915/    2FAE :
7916/    2FAE : CC 06               ShiftAndXor     ld R12, #6              ; shift 6 bytes
7917/    2FB0 : DC 11                               ld R13, #Wrk_Sys+1      ; start with R1
7918/    2FB2 : CF                                  rcf
7919/    2FB3 :                     ;
7920/    2FB3 : C1 ED               S_A_Xor_Lp      rrc @R13
7921/    2FB5 : DE                                  inc R13
7922/    2FB6 : CA FB                               djnz R12, S_A_Xor_Lp
7923/    2FB8 :                     ;
7924/    2FB8 : FB 12                               jr NC, S_A_Xor_End
7925/    2FBA : B6 E1 8C                            xor R1, #140
7926/    2FBD : B6 E2 0C                            xor R2, #12
7927/    2FC0 : B6 E3 0A                            xor R3, #10
7928/    2FC3 : B6 E4 28                            xor R4, #40
7929/    2FC6 : B6 E5 18                            xor R5, #24
7930/    2FC9 : B6 E6 08                            xor R6, #8
7931/    2FCC : 42 11               S_A_Xor_End     or R1, R1               ; If R1=0 ...
7932/    2FCE : AF                                  ret
7933/    2FCF :
7934/    2FCF :
7935/    2FCF :                     ;*****************************************************************
7936/    2FCF :                     ;
7937/    2FCF :                     ; Procedure: TestMod8
7938/    2FCF :                     ;
7939/    2FCF :                     ;  This procedure is used to test if J mod 8 = 0.
7940/    2FCF :                     ;
7941/    2FCF :                     ; Inputs: J: WORD {RR8}
7942/    2FCF :                     ;
7943/    2FCF :                     ; Outputs: none
7944/    2FCF :                     ;
7945/    2FCF :                     ; Global Variables Changed: Done:        BOOLEAN {R7/bit 5}
7946/    2FCF :                     ;                           Correctable: BOOLEAN {R7/bit 7}
7947/    2FCF :                     ;
7948/    2FCF :                     ; Algorithm:
7949/    2FCF :                     ;
7950/    2FCF :                     ;  Begin
7951/    2FCF :                     ;    If J mod 8 = 0
7952/    2FCF :                     ;      Then
7953/    2FCF :                     ;        Done:=true
7954/    2FCF :                     ;        Correctable:=true
7955/    2FCF :                     ;  End
7956/    2FCF :                     ;
7957/    2FCF :                     ;*****************************************************************
7958/    2FCF :
7959/    2FCF : 08 E9               TestMod8        ld R0, R9
7960/    2FD1 : 56 E0 07                            and R0, #7              ; get remainder from division
7961/    2FD4 : EB 03                               jr NZ, TstMd8_Done
7962/    2FD6 : 46 E7 A0                            or R7, #Ecc_Done+Ecc_Correctable
7963/    2FD9 : AF                  TstMd8_Done     ret
7964/    2FDA :
7965/    2FDA :
7966/    2FDA :                     ;*****************************************************************
7967/    2FDA :                     ;
7968/    2FDA :                     ; Procedure: Test0
7969/    2FDA :                     ;
7970/    2FDA :                     ;  This procedure is used to test if J=0.
7971/    2FDA :                     ;
7972/    2FDA :                     ; Inputs: J: WORD {RR8}
7973/    2FDA :                     ;
7974/    2FDA :                     ; Outputs: none
7975/    2FDA :                     ;
7976/    2FDA :                     ; Global Variables Changed: Done:        BOOLEAN {R7/bit 5}
7977/    2FDA :                     ;                           Correctable: BOOLEAN {R7/bit 7}
7978/    2FDA :                     ;
7979/    2FDA :                     ; Algorithm:
7980/    2FDA :                     ;
7981/    2FDA :                     ;  Begin
7982/    2FDA :                     ;    If J=0
7983/    2FDA :                     ;      Then
7984/    2FDA :                     ;        Done:=true
7985/    2FDA :                     ;        Correctable:=true
7986/    2FDA :                     ;  End
7987/    2FDA :                     ;
7988/    2FDA :                     ;*****************************************************************
7989/    2FDA :
7990/    2FDA : 08 E8               Test0           ld R0, R8
7991/    2FDC : 42 09                               or R0, R9               ; If J=0 ...
7992/    2FDE : EB 06                               jr NZ, Test0_Done
7993/    2FE0 : 46 E7 20                            or R7, #Ecc_Done
7994/    2FE3 : 56 E7 7F                            and R7, #0FFh-Ecc_Correctable
7995/    2FE6 : AF                  Test0_Done      ret
7996/    2FE7 :
7997/    2FE7 :
7998/    1FE7 :                             dephase
7999/    1FE7 : =>FALSE                     if $>ROMsize
8000/    1FE7 :                                     error "\aROM size exceeded !!!"
```

```
8001/    1FE7 : =>TRUE                      elseif
8002/    1FE7 : FF FF FF FF FF FF                   DB ROMsize-$ dup(0FFh)  ; pad with $FF's
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF
8003/    2000 : [7999]                      endif
8004/    2000 :                             end
```

symbol table (* = unused):

ABORT		51F	-	ABORT_STAT	16D8 C
ACCESS		80	-	ACCESS_OFFSET	90 -
ACERASEL		20	-	ACK_READ	302 -
*ADD3		3AC	-	ADDSPARE	1568 C
ADS_ELSE1		1576	C	ADS_UPDATE	1586 C
*APL_ACK		55	-	*APL_EXCPT	0 -
*APPLE_MEM		0	-	*ARCHITECTURE	i386-unknown-win32 -
AUTO_OFFSET		279F	C	*AUTO_OFF_RET	27E9 C
*B0_7_IO		0	-	*B0_7_SER	40 -
*B0_VCTTAB		100A	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	1546 C
*BAD_55		80	-	*BAD_CMND	1 -
*BAD_PARAMS		10	-	*BAD_PASSWORD	F -
BAD_RCVR_RET		1785	C	BAD_RC_ECC	177E C
BAD_RC_SET		1776	C	BAD_RC_SPR	177A C
BAD_RECOVER		1758	C	*BAD_STATE	0 -
*BANKREG		1800	-	BANK_CALL	4AB -
BANK_RET		4F8	-	*BIGENDIAN	0 -
*BLKLNDEX		1752	-	BLKOFFSET	1751 -
BLKSTAT		5A	-	BLK_MOVE	2156 C
BLOCKID		200	-	BLOCKLENGTH	21D -
BLOCKMOVE		214F	C	*BRANCHEXT	0 -
BSY		8	-	BUF2ARRAY	1274 C
*BUF2CRC		1489	C	*BUF2ECC	148B C
*BUF2PW2		1491	C	BUF2_TO_RBUF	205B C
BUF2_TO_WRBUF		2070	C	*BUFDUMMY	1274 C
*BUFFER2		1275	C	BUF_DAMAGE	20 -
B_BLOCK		2	-	B_MOVE	20A6 C
B_RCVR_READ		1760	C	CACHEARRAY	16FC C
CACHELENGTH		14	-	CACHESTAT	16E8 C
CACHE_CNVRT		2C2F	C	CACHE_INDEX	5B -
CACHE_NOSPARE		2B92	C	CACHHDCHG	40 -
CACHSEEK		80	-	CALCMAGDIR	2728 C
*CASESENSITIVE		0	-	CHECKSUM0	D47C -
CHECKSUM1		EF78	-	*CHKB_MISMATCH	0 -
CHKOFF_NOOFF		27FC	C	CHK_CHAIN	15DA C
CHK_CHK_BYTE		61D	-	CHK_FATALSTAT	1CD9 C
CHK_FMTPARMS		1BEB	C	CHK_HDPTR	15B8 C
CHK_INST		1204	C	CHK_LIMITS	1216 C
CHK_MVWRDATA		278F	C	CHK_OFFSET	27F7 C
CHK_OFF_RET		2804	C	CHK_PARK	2E72 C
CHK_PARK1		421	-	*CHK_PARK3	2E81 C
CHK_PASSWORD		24E5	C	CHK_PK_INT	2E7E C
CHK_PK_JP		2E7A	C	CHK_P_END	24FF C
CHK_P_LP		24ED	C	CHK_SPR2	2416 C
CHK_SPRCHK		2405	C	CHK_SPRCNT	254B C
CHK_SPRTBL		236D	C	CHK_SPR_END	2424 C
CHK_SPR_RET		2562	C	CHK_SSTAT	2708 C
CLEARBITMAP		20	-	CLEARSTATUS	63D -
CLRNMSTAT3		1E97	C	CLRNORMSTAT	1E5F C
*CLRSTAT_VECTOR		1039	C	CLR_BSY	25F -
CLR_DMT		41F	-	CLR_N_PWRRST	1E7C C
CLR_N_SLFTST		1E8B	C	CLR_N_STAT	1E67 C
*CMD		4	-	CMNDTYPE	F0 -
*CMND_EXCPT		1	-	*CMND_LEN	7 -
CMND_LIMITS		124A	C	CMND_PENDING	80 -
CMND_PTR		101A	-	CMND_PTRS	124D C
CNVRTLOGICAL		163A	C	*CNVRTSRCH	163A C
*COMM_ERR		1	-	*CONSTPI	3.141592653589793 -
CRCERRL		80	-	CRCSTAT	40 -
CREATE_TBL		22D8	C	CSTATUS0	1495 C
*CSTATUS1		1499	C	*CSTATUS2	149D C
CSTATUS3		14A1	C	CSTATUS4	14A5 C
*CSTATUS5		14A9	C	*CSTATUS6	14AD C
*CS_VECTOR		1016	C	CUR_CYL	50 -
CUR_THS		1754	-	CYLINDER	52 -
C_MAGDIR_ELSE		2744	C	C_MAGDIR_RET	2757 C
DATARECAL		40	-	DATA_EX_ABORT	16DD C
DATA_EX_CASE		16C7	C	DATA_EX_CMND	16E0 C
DATA_EX_HANDLER		16B1	C	DATA_EX_RDERR	16C4 C
DATA_TYPE		58	-	*DATE	7/31/2013 -
DEC_BADCNT		2	-	DELETESPARE	15AB C
DEVICEPARAMS		10A7	C	DEVSUBTYPE	0 -
DEV_PARM_LENGTH		1E	-	*DIAGNOSTIC	20 -
DIAG_CMNDS		13	-	DIAG_INST	1259 C
DIR_FRWD		20	-	DISKSTAT	56 -
*DISK_MEM		20	-	DISK_SPEED	20 -
DIV3_19		257B	C	DIV3_19_DIV2	25B7 C
DIV3_19_LP		2596	C	DIV3_38	2570 C
*DIV3_76		2565	C	DIV3_X	2584 C
DIVHDSSCTRS		25D4	C	DIVSCTRS	25E0 C
*DM		10	-	*DMT_C00	1 -
*DMT_C01		2	-	*DMT_C02	3 -
*DMT_COUNTER		36	-	DMT_FMTTRACK	8 -
*DMT_FORMATBLOCK		7	-	DMT_LCTSCTR	9 -
*DMT_OVERLAP		D	-	*DMT_POSHEADS	0 -
*DMT_RD_COMMON		E	-	*DMT_READBLOCK	4 -
*DMT_READHDR		5	-	*DMT_RECAL	A -
*DMT_SEEK		F	-	*DMT_SERVOOK	10 -
*DMT_S_CMND		13	-	*DMT_S_LOAD	16 -
*DMT_S_R		C	-	*DMT_S_STAT	14 -
*DMT_S_STORE		15	-	*DMT_TRK_CNT	12 -
*DMT_VAL		1F4	-	*DMT_WRITEBLOCK	6 -
*DMT_WR_COMMON		11	-	*DMT_WV	B -
*DM_MASK		8	-	DONTLISTINCLS	1 -
DO_READ		1EE7	C	DO_WRITE	1DFA C
*DRWL_READ		80	-	*DRWL_WRITE	0 -
D_CHK_ELSE		15FA	C	D_INIT_SPRTBL	1C48 C
D_I_SPRTBL		1C63	C	D_RDHDR_RESP	C -
D_RDH_CRC		19E5	C	D_RDH_END	19F3 C
D_READ		1999	C	D_READHDR	19B5 C
D_READ_END		19B2	C	D_READ_ID	1818 C
D_READ_RESPONSE		B	-	D_READ_SPRTBL	1867 C
D_RSTSRVO		1C35	C	D_R_ID_RESPONSE	2 -
D_R_SPR_RESP		F	-	D_SCAN	2CC1 C
D_SCAN_RESPONSE		15	-	D_SCTRS1	25EC C
D_SCTRS2		25EE	C	D_WRITE	1A0B C
D_WRITE_END		1A24	C	D_WRITE_RESP	D -
ECC		2F06	C	*ECCERROR	0 -

```
 ECCSTAT :                 80 - | *ECCTEST :                   2 - |
 ECC_ALIGN :            2F37 C |  ECC_ALIGNED :              40 - |
 ECC_AL_1 :             2F51 C |  ECC_AL_2 :               2F59 C |
 ECC_CORRECTABLE :         80 - |  ECC_CRCT :               2F7B C |
 ECC_CRCT_LP :          2F97 C |  ECC_DIV8 :               2F82 C |
 ECC_DONE :                20 - |  ECC_END :                2FAB C |
 ECC_LD_LP :            2F16 C |  ECC_LHJ_WHILE :          2F1F C |
 ECC_SHFT_2 :           2F72 C |  ECC_SHFT_ELSE :          2F6F C |
*END_WR_RESPONSE :          6 - | *END_CSTATUS :            14B1 - |
*EPPOM0 :                   0 - |  ENTER_CACHE :            2BF7 C |
 EPROM2 :                   2 - | *EPROM1 :                   1 - |
*EPROM4 :                   4 - | *EPROM3 :                   3 - |
 EPROMSIZE :            1000 - | *EPROMOFFSET :            1000 - |
 EPROMTEST :             189 - | *EPROMSTARTADR :            1 - |
 ERROR :                   80 - |  EPROM_FAIL :              40 - |
 EXCPT_STAT :              24 - |  EXCEPT_RETURN :          1727 C |
 EXTPUSH_VECTOR :       1097 C |  EXTPOP_VECTOR :          109F C |
*EXT_CLK :                  0 - | *EXTSTK_VECTOR :          1029 C |
 EXT_POP_LP :           2244 C |  EXT_POP :                2236 C |
 EXT_PUSH_LP :          2224 C |  EXT_PUSH :               2215 C |
 EX_CASE_MAX :               A - |  EX_BADBLOCK :              4 - |
 EX_HDRSPR :                 A - |  EX_HDRBAD :                8 - |
 EX_READERR :                6 - |  EX_JP_TBL :              16F1 C |
*EX_UNDETERMINED :           0 - |  EX_SPRBLOCK :              2 - |
*FBUF1CRC :              1266 C | *FALSE :                    0 - |
 FBUFFER1 :             1052 C | *FBUF1ECC :               1268 C |
*FDATASYNC :            1050 C | *FDATAGAP :               1042 C |
*FHDRGAP :              102A C |  FENDGAP :                126E C |
*FHEADER :              103C C | *FHDRSYNC :               103A C |
*FMENL :                    0 - |  FLAGS :                   FC - |
*FMTDELAY :             1007 C | *FMTBLK_ABORT :             7 - |
*FMTLNTRL :             14C4 C | *FMTERROR :                80 - |
*FMTSRVOERR :             40 - |  FMTOFFSET :              14C3 C |
 FMTT2_UNTIL :          111F C | *FMTSUCCESS :              20 - |
*FMTTRK_ABORT :             8 - |  FMTTRK_2 :               1108 C |
 FMTT_1 :               10D6 C | *FMTTRK_POSERR :            0 - |
 FMTT_3 :               10E3 C |  FMTT_2 :                 10DE C |
 FMT_BEGIN :            10F9 C |  FMTT_UNTIL :             1114 C |
 FMT_OFF1 :             10EF C |  FMT_CHK_INTER :          1C0F C |
 FMT_RESPONSE :            11 - |  FMT_OFF2 :               10F2 C |
 FORMAT1 :              1C00 C |  FORMAT :                 1BD4 C |
 FORMATBLOCK :           446 - | *FORMATARRAY :            1020 C |
 FORMAT_ABORT :         1C08 C |  FORMATTRACK :            10CC C |
 FREEP_NOPARK :         2E8B C |  FOUND :                    1 - |
 FREE_EPROM :           2EC8 C |  FREEP_SLFTST :           2E94 C |
*FREE_PROC :              69 - |  FREE_MTRSPD :            2EE2 C |
 FREE_RW :              2EF1 C |  FREE_RAM :               2EBA C |
 FREE_SLFTST :             23 - |  FREE_SCTRCNT :           2ED4 C |
 FRMTRECAL :               70 - | *FREE_VECTOR :            1021 C |
*FSCTRGAP :             1020 C | *FRST_SPRTBL :              80 - |
*GATE_CLK :               10 - | *FULLPMMU :                 1 - |
 GDHDR_2 :              2132 C |  GDHDR_1 :                20FC C |
 GDHDR_LP :             2107 C |  GDHDR_END :              2149 C |
 GETNEWSPARE :          14FB C |  GEN_CHK_BYTE :            62E - |
 GET_EL_DONE :          17BD C |  GET_CYL_H_S :            20D4 C |
 GET_HEADPTR :          1786 C |  GET_EOLIST :             17A3 C |
 GET_PTR :              17F8 C |  GET_PREVIOUS :           1604 C |
 GET_TYPE :             219B C |  GET_SPR_CODE :           1812 C |
 GET_TYPE_END :         21E7 C |  GET_TYPE_CHECK :         21D2 C |
 GET_ZONE :             2807 C |  GET_WR_DATA :             2ED - |
 GNS_END :              155E C |  GNS_CHK_HI :             1542 C |
 GNS_LP1END :           1521 C |  GNS_LP1 :                150F C |
 GNS_LP2END :           1533 C |  GNS_LP2 :                1523 C |
 GS_LDAO1 :             26B9 C |  GOODHDR :                20FA C |
 GTSK_LDAO :            26B6 C |  GS_LDNO1 :               26C1 C |
 GTZN_LP :              280E C |  GTSK_LDNO :              26BE C |
 GT_3_LDEI :            2C54 C |  GTZN_PUSH :              2828 C |
 GT_PBLOCK :            2E35 C |  GT_BLK :                 2C7F C |
 G_T_CHKID :            21BF C |  GT_THS :                 2C4C C |
*HAS64 :                    1 - |  G_T_PROCNVRT :           21B7 C |
*HASFPU :                   0 - | *HASDSP :                   0 - |
 HDR_MISMATCH :            20 - | *HASPMMU :                  0 - |
*HD_DIR_REV :               0 - | *HD_DIR_FRWD :              4 - |
 HEAP :                 174E - |  HEAD :                    54 - |
 HIMAXLOGICAL :              0 - |  HIMAXCYL :                 2 - |
*HIREGADR :                7F - | *HIRAMADR :                7FF - |
 HISPR0 :                   0 - |  HIREVNUMBER :              1A - |
*HI_RWI_REG :              20 - |  HISPR1 :                   0 - |
*HOST_OVRFLOW :            12 - | *HOME :                    C0 - |
 IBSY :                    40 - |  HS0 :                     20 - |
 ILLEGAL_BLOCK :           40 - |  ID_TYPE :                  4 - |
 INC_BADCNT :               1 - | *IMR :                     FB - |
 INC_SPRCNT :               0 - |  INC_BLK :                2C5B C |
*INEXTMODE :                0 - |  INDEXMARK :                4 - |
 INIT_HICYL :               1 - |  INIT_EXTSTACK :          2204 C |
*INIT_PC :                  C - |  INIT_LOCYL :              F9 - |
 INIT_SPRTBL :          229F C |  INIT_RESPONSE :            1 - |
*INMAXMODE :                0 - | *INLWORDMODE :              0 - |
 INST_ABORT :           1227 C | *INSRCMODE :                0 - |
 INTERTABLE :           10C5 C | *INSUPMODE :                0 - |
*INT_OUT :                 C0 - | *INTL_DFLT :                2 - |
 IRQ :                     FA - | *IPR :                     F9 - |
 IRQ_SECDN :                2 - |  IRQ_INDEX :                1 - |
 ISCAN_RET :            107C C |  IRQ_SECTOR :               4 - |
*ISCAN_VECTOR :         1063 C |  ISCAN_SPRCHK :           1075 C |
 I_MAP_LP2 :            230E C |  I_MAP_LP :               22FF C |
 I_SPR_RESPONSE :           12 - | *I_OREGUSED :               4 - |
 I_S_TBL_LP4 :          22C9 C |  I_S_TBL_LP2 :            22BD C |
*LBLK_BOUNDS :              2 - |  K1 :                     10BF - |
 LCTSCTR1 :             114C C |  LCTDONE :                1182 C |
 LCTSCTR3 :             118A C |  LCTSCTR2 :               1151 C |
 LCTSCTR5 :             116B C |  LCTSCTR4 :               1162 C |
 LCT_OFFSET :           113A C |  LCTSCTR6 :               117F C |
*LDPW_VECTOR :          1019 C | *LC_VECTOR :              1051 C |
 LD_BLK_BB :            2C01 C |  LD_BLKSTAT :             2C04 C |
 LD_CACHE_IN :          2B8D C | *LD_CACHE_END :           2BF2 C |
 LD_CACHE_MORE :        2BF0 C |  LD_CACHE_LP :            2B76 C |
 LD_CACH_SEEK :         2BDC C |  LD_CACH_HDCHG :          2BCC C |
 LD_C_LONG :            2CB8 C |  LD_C_ENTER :             2BEB C |
 LD_C_S_END :           2CB5 C |  LD_C_SRCH :              2C83 C |
 LD_LGCLBLK :            64E - |  LD_C_S_RET :             2CB7 C |
 LD_PARAM1 :            1AE3 C | *LD_OFF_VAL :               20 - |
 LD_STAT1 :             1A60 C |  LD_STAND_STAT :          12E6 C |
 LD_STAT2_1 :           1A6D C |  LD_STAT2 :               1A6B C |
 LD_STAT3 :             1A75 C |  LD_STAT2_LP :            1A6F C |
 LD_STAT5 :             1A83 C |  LD_STAT4 :               1A79 C |
 LD_STAT7 :             1A9B C |  LD_STAT6 :               1A91 C |
                               |  LD_STAT_END :            1AAB C |
```

```
 LD_STAT_REG :                1AA3 C |  LD_S_CMND_LP :                2983 C |
 LD_TMSTMP :                  23DC C |  LD_TM_LP :                   23DE C |
*LED :                        1800 - | *LEDSTAT :                      1 - |
*LED_MASK :                     FE - | *LH_VECTOR :                 1010 C |
*LISTON :                        1 - | *LL_VECTOR :                 100D C |
 LOADSTATUS :                  3FC - |  LOAD_CACHE :                2B41 C |
 LOAD_HEADER :                 5E1 - |  LOAD_LOGICAL :               60D - |
 LOAD_PASSWORD :              2506 C |  LOAD_SPRTBL :               231D C |
 LOAD_SRVOCMND :              297B C |  LOAD_TOS :                  2252 C |
 LOCATESECTOR :               1124 C |  LOGICALBLOCK :              14A1 - |
 LOMAXCYL :                      20 - |  LOMAXLOGICAL :                FF - |
 LOREVNUMBER :                   45 - |  LOSPR0 :                      55 - |
 LOSPR1 :                        AA - | *LO_RWI_REG :                  21 - |
 LPW_LP :                     250C C |  LST_HEAD :                    5E - |
 LST_HICYL :                     5C - |  LST_LOCYL :                   5D - |
 LST_SECTOR :                    5F - |  L_RD_LP :                   233C C |
 L_SPRTBL_LP :                2324 C |  L_SPRTBL_MORE :             2360 C |
 L_SPR_END :                  23CC C |  L_SPR_INC :                 23C6 C |
 L_SPR_MOVE :                 23C3 C | *MACEXP :                       1 - |
 MAP_DFLT :                       C - |  MAP_TABLE :                 1681 C |
*MAXCHKBAD :                      A - | *MAXEPROMADDRESS :            FFF - |
*MAXSEEK :                      185 - |  MAX_CMND_TYPES :              2 - |
 MAX_INTERLEAVE :                 6 - | *MEM_EXT :                     20 - |
*MEM_NORM :                       0 - |  MIDMAXLOGICAL :               4B - |
 MIDSPR0 :                       19 - |  MIDSPR1 :                     32 - |
*MID_CYI :                      101 - |  MINOFFSET :                    A - |
*MOD_N :                          1 - |  MOMCPU :                    8601 - |
*MOMCPUNAME :                  Z8601 - | *MOVE4 :                     21EA C |
*MOVE4C :                     21F7 C |  MOVE4C_LP :                 21F9 C |
 MOVE4_B0 :                   1E9A C |  MOVE4_B0_LP :               1E9C C |
 MOVE4_LP :                   21EC C |  MOVEL_3 :                   185E C |
 MOVEL_3_LP :                 1862 C | *MSEL0 :                        10 - |
*MSEL1 :                         20 - |  MSWAIT :                     1CB - |
 MTRSPD :                     2A6D C |  MTRSPD_END :                2A97 C |
 MTRSPD_IDX :                 2A85 C |  MTRSPD_LP1 :                2A7A C |
 MTRSPD_LP2 :                 2A8C C |  MTRSPD_LV :                 2AAE C |
 MULR0_M :                    25CB C | *MULTIBLK :                     4 - |
 MULTIWR :                       20 - |  MVWR_END :                  279E C |
 NBRHDS :                         2 - |  NBRSCTRS :                   13 - |
 NBRTRACKS :                    202 - | *NBRZONES :                    10 - |
*NESTMAX :                      100 - |  NEW_SEEK :                   5D7 - |
 NIL :                           80 - | *NOHDR_STATE :                  0 - |
*NON_RETRIG :                    20 - | *NORMFMT_STATE :                A - |
 NORM_STATE :                     2 - | *NOSPACE :                      0 - |
 NOTINTABL :                  14D2 C | *NOT_ACERASEL :                 0 - |
*NOT_BSY :                        0 - | *NOT_ECCERROR :                 2 - |
*NOT_FMENL :                     20 - | *NOT_RDHDRH :                   0 - |
 NOT_SERVORST :                   10 - | *NOT_STARTL :                   1 - |
*NOT_Z8TESTL :                    40 - |  NO_SPRTBL :                    1 - |
 NZERO_STAT :                      8 - |  OFFSET :                      10 - |
 OFFSET_ON :                       1 - |  OFF_AUTO :                    40 - |
 OFF_DIR_FRWD :                   80 - |  OFF_DIR_REV :                  0 - |
 OK_INST :                    122A C |  ON_TRACK :                    80 - |
*OPEN_DRAIN :                     0 - |  OP_FAILED :                    1 - |
 OVERLAP :                    1F8D C |  OVERLD_SCTR :               1FB3 C |
 OVRLAPSEEK :                 1FBC C | *OVRLP_2 :                   1FCC C |
*OVRLP_ABORT :                    B - |  OVRLP_END :                 1FE4 C |
*OVRLP_LP :                   1FC3 C | *OVRLP_S_1 :                 1FDE C |
*OVR_LDCACHE :                1FA5 C |  P0 :                           0 - |
*P01M :                          F8 - | *P01M_IMAGE :                   4 - |
*P01M_STMACH :                    6 - | *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 - | *P21_IN :                       2 - |
*P22_IN :                         4 - | *P26_IN :                      40 - |
*P2M :                           F6 - |  P3 :                           3 - |
*P3M :                           F7 - | *P3M_IMAGE :                    5 - |
*P3M_STMACH :                     7 - | *PACKING :                      0 - |
*PADDING :                        1 - |  PARKCYL :                    235 - |
 PARKED :                        10 - |  PARK_HEADS :                2988 C |
*PAR_OFF :                        0 - | *PAR_ON :                      80 - |
 PASSWORD :                   1003 C |  PASSWRD1 :                  F078 - |
 PASSWRD2 :                   3C1E - |  PBLK :                      1758 - |
 PBLOCK :                     174E - | *PC :                           8 - |
*POSHDS_3 :                   26EA C |  POSHDS_4 :                  26DE C |
 POSHDS_42 :                  26E2 C |  POSHDS_5 :                  26FB C |
 POSHDS_6 :                   26FF C |  POSHEADS :                  2688 C |
 POWER_RESET :                   80 - |  PRE0 :                        F5 - |
*PRE1 :                          F3 - |  PROFILE :                      0 - |
 PRORD_ERR :                  12C7 C |  PROWR_1 :                   188A C |
*PROWR_EXIT :                 188D C |  PRO_CMNDS :                    2 - |
 PRO_INST :                   1253 C |  PRO_LOG_OFFSET :               1 - |
 PRO_RD_BB :                  12DB C | *PRO_RD_EXIT :               12BD C |
 PRO_READ :                   1287 C |  PRO_WRITE :                 187A C |
 PRO_WRVER :                  1942 C |  PRO_WRV_1 :                 1959 C |
*PRO_WRV_2 :                  1945 C | *PRO_WR_BB :                 1885 C |
 PRO_WR_SETUP :               1895 C |  PSECTOR :                     22 - |
*PWRFLG0 :                        2C - | *PWRFLG1 :                     2D - |
*PWRFLG2 :                        2E - | *PWRFLG3 :                     2F - |
 PWRRST :                        10 - | *PWR_ON_RESET :                40 - |
 P_W_S_END :                  18C1 C | *R2MASK :                       0 - |
 R3MASK :                         F - | *RAM0 :                         0 - |
*RAM1 :                           1 - | *RAM2 :                         2 - |
*RAM3 :                           3 - | *RAMBANK0 :                  1900 - |
*RAMLPTIMES :                     2 - | *RAMOFFSET :                 1000 - |
 RAMSIZE :                      800 - |  RAM_FAIL :                    80 - |
*RANDOM :                        80 - | *RBUF1CRC :                  122D C |
 RBUF1ECC :                   122F C | *RBUF1PW :                   1236 C |
 RBUFFER1 :                   1019 C |  RBUF_TO_BUF2 :              2099 C |
 RBUF_TO_SPR :                2089 C |  RCVR_SPRTBL :               24B3 C |
*RDATAGAP :                   1011 C |  RDBLK_ABNORM :              1F34 C |
*RDBLK_ABORT :                    4 - |  RDBLK_END :                 1F20 C |
 RDBLK_NOHDR :                1F3C C | *RDBLK_NORM :                1EF8 C |
 RDBLK_RMOVE :                1F84 C |  RDBLK_RPT :                 1EB2 C |
 RDBLK_UNTIL :                1F16 C |  RDBLK_VECTOR :              1083 C |
 RDB_RPT1 :                   1F2A C |  RDCRCERR :                     8 - |
 RDERRCNT :                      27 - |  RDERROR :                     80 - |
*RDHDRH :                        80 - |  RDHDRRECAL :                  40 - |
*RDHDR_ABORT :                    5 - |  RDHDR_NORM :                2026 C |
 RDHDR_RESIDENT :               340 - |  RDHD_SERVOERR :             2032 C |
 RDHD_SRVOOK :                2037 C |  RDHERROR :                    80 - |
 RDHSRVOERR :                    40 - |  RDH_CHKECC :                203C C |
 RDH_CRCERR :                 2049 C |  RDH_END :                   2051 C |
 RDH_STAT_ARRAY :             1008 - | *RDL_VECTOR :                1013 C |
 RDNOHDRFND :                    10 - |  RDSRVOERR :                   40 - |
```

```
 RDSTAT :                    26 - |  RDSUCCESS :                  20 - |
 RDUMMY :                  1018 C |  RD_ABRT_RESP :               13 - |
 RD_ABT_LP :               1C28 C |  RD_BAD1 :                  1F6A C |
 RD_BADBLOCK :             1349 C |  RD_BADCRC :                1F5E C |
 RD_BADECC :               1F7E C |  RD_B_CRC_1 :               1F7A C |
*RD_CMN_ABORT :               C - |  RD_CMN_CRCT :              1332 C |
 RD_CMN_END :              134B C |  RD_CMN_HDR :               1329 C |
 RD_CMN_LP :               1305 C |  RD_CMN_RDERR :             1345 C |
*RD_CMN_SERVO :            131A C |  RD_HDRERR :                1F54 C |
 RD_LEAVE :                12CA C |  RD_LEAVE1 :                12D2 C |
 RD_LEAVE2 :               12CF C | *RD_NOCRCERR :              1F11 C |
 RD_RESIDENT :              346 - |  RD_SERVOERR :              1F59 C |
*RD_SERVOOK :              1F07 C |  RD_SPRBLOCK :              1341 C |
 RD_SSTAT_LP :             1AF0 C |  RD_SSTAT_RESP :               4 - |
 RD_STAT_RESP :               3 - |  READARRAY :                1000 C |
 READBLOCK :               1EA9 C |  READHDR :                  2007 C |
*READHDRARRAY :            1000 C |  READSTATUS :                 0 - |
 READ_ABORT :              1C15 C |  READ_COMMON :              12F2 C |
 READ_CSTATUS :            1A27 C |  READ_FAST :                1EC9 C |
 READ_ID :                 1821 C |  READ_ID_LP :               1832 C |
*READ_OP :                   80 - |  READ_RESPONSE :              2 - |
 READ_SPRTBL :             1870 C |  READ_SSTATUS :             1AB3 C |
 RECOVERY :                  80 - |  REGCOUNT :                  80 - |
*REGLPTIMES :                 2 - | *REGUSED :                    5 - |
*RELAXED :                    0 - |  REMAP_SECTOR :             25EF C |
*RENDGAP :                 1235 C |  RESEEK :                    5CA - |
 RESETSERVO :              29A2 C |  RESET_STMACH :              5AE - |
 RESTORE :                 28E2 C |  RESTORE_END :              2942 C |
 RESTORE_LP :              291B C |  REST_UP1 :                 2940 C |
 REST_UP2 :                2917 C |  REST_UPDATE :              292B C |
*RETRIG :                    30 - |  RETURN_VECTOR :            20A9 C |
*RHBUF1CRC :               122D C | *RHBUF1ECC :                122F C |
*RHBUF1PW :                1236 C | *RHBUFFER1 :                1019 C |
*RHDATAGAP :               1012 C | *RHDRGAP :                  100A C |
*RHDUMMY :                 1018 C |  RHEADER :                  100B C |
*RHENDGAP :                1235 C | *RHHDRGAP :                 100A C |
 RHHEADER :                100C C | *RHSCTRGAP :                1000 C |
*ROMBANK0 :                1E00 - | *ROMBANK2 :                 1D00 - |
 ROMSIZE :                 2000 - |  RP :                         FD - |
*RSCTRGAP :                1000 C |  RSTSRVO_RESP :              14 - |
*RWI :                        4 - |  RWI_CYLINDER :             101 - |
*RWI_VALUE :               1008 C |  RWTEST :                   2AE9 C |
 RWTEST1 :                 2AF5 C |  RWTEST_END :               2B3B C |
 RWTEST_LP :               2B02 C |  RW_FAIL :                    2 - |
 RW_NEXT :                 2B37 C |  SAVE_PREVIOUS :            160A C |
 SCAN :                    2CCA C |  SCAN_1 :                   2CD2 C |
 SCAN_CLR :                2E13 C | *SCAN_CNVRT :               2CF4 C |
*SCAN_END :                2D9B C | *SCAN_EXCEPT :              2CEF C |
 SCAN_INCTRKS :            2D80 C |  SCAN_LB_LP :               2E23 C |
 SCAN_LDLB :               2E1B C |  SCAN_LD_L :                2D2C C |
 SCAN_LP :                 2CE0 C |  SCAN_MAP_END :             2D0B C |
 SCAN_MAP_LP :             2CFC C |  SCAN_MORE :                2D58 C |
 SCAN_M_POP :              2D55 C | *SCAN_RD_ERR :              2D4B C |
 SCAN_READ :               2CE6 C |  SCAN_VECTOR :              1059 C |
 SCN_BADLP :               2D9D C | *SCN_BD1 :                  2DAD C |
 SCN_BD2 :                 2DFA C |  SCN_BD3 :                  2DA3 C |
 SCN_BDNEXT :              2E04 C |  SCN_INIT1 :                2CD8 C |
 SCN_SPARE :               2D32 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 - |  SCR_CNTR :                   38 - |
 SCTRCOUNT :               2AC8 C |  SC_VECTOR :                1088 C |
 SD_S_C_RESP :                5 - | *SECTDNL :                    8 - |
 SECTOR :                    55 - | *SECTORMARK :                 2 - |
 SECTOR_CNT :                 8 - |  SEEK :                     2605 C |
 SEEKCOMPLETE :               2 - |  SEEKCOUNT :                  2A - |
*SEEK_ABORT :                 D - |  SEEK_ABT :                 263D C |
 SEEK_END :                264F C |  SEEK_LP :                  261E C |
*SEEK_RET :                2662 C | *SEEK_RETRY :               261C C |
 SEEK_TYPE :                 57 - |  SEEK_VECTOR :              108D C |
 SEEK_WAIT :               264A C |  SEEK_WT :                  266C C |
 SEEK_WT_LP :              266F C |  SEGPTRARRAY :              14C5 C |
 SELECTHEAD :              2717 C |  SELFTEST :                 2A22 C |
 SELHD_END :               2723 C | *SELHD_RET :                2725 C |
 SEL_HEAD0 :               2720 C |  SEND_PARK :                1B82 C |
 SEND_RESTORE :            1B46 C |  SEND_SEEK :                1B22 C |
 SEND_SERVOCMND :          1AF7 C |  SEQ_CACHSRCH :               8 - |
 SERIAL_IN :                  8 - |  SERIAL_OUT :                10 - |
*SERR_NOTREADY :              0 - |  SERVOCMND :                2837 C |
 SERVOERR :                  10 - |  SERVOLOAD :                28BC C |
 SERVOOK :                 275A C |  SERVORDY :                   20 - |
*SERVORST :                   0 - |  SERVOSTATUS :              285D C |
 SERVOSTORE :              287F C | *SERVO_DEAD :                 0 - |
 SERVO_FAIL :                10 - | *SERVO_LEN :                  5 - |
 SERVO_STLP1 :             2861 C |  SETBITMAP :                  40 - |
 SETSTATUS :                403 - |  SET_AUTOOFFSET :           1B95 C |
 SET_DMT :                  413 - |  SET_RCVR_RESP :              8 - |
 SET_RCVR_STORE :          1B7F C |  SET_RECOVERY :             1B69 C |
 SET_SEEKNEEDED :          2A0E C |  SHIFTANDXOR :              2FAE C |
*SINGLE_PASS :                0 - |  SIO :                        F0 - |
 SIORDY :                    40 - |  SLFTST_RESULT :              25 - |
 SLFTST_TABLE :            2EA8 C | *SLFTST_VECTOR :            1041 C |
 SLF_LEAVE :               2A6A C | *SLF_RWTEST :               2A62 C |
*SLF_SECTORS :             2A4F C | *SLF_STATE :                2A57 C |
*SLF_TRACKS :              2A49 C |  SOK_CHKRDY :               2774 C |
*SOK_PARK :                2779 C |  SPARE :                      10 - |
*SPAREABORT :                 9 - |  SPAREARRAY :               14BB C |
 SPAREBITMAP :             1547 C |  SPAREBLOCK :               13AE C |
 SPARECHECK :              1694 C |  SPARECOUNT :               2519 C |
 SPAREEND :                169A C |  SPARELENGTH :               1DF - |
 SPAREPW1 :                14BB C |  SPAREPW2 :                 1696 C |
 SPARETABLE :              1551 C |  SPARETMSTMP :              14BF C |
 SPH :                       FE - |  SPL :                        FF - |
 SPR :                     2429 C |  SPRBLK_1 :                 139C C |
 SPRBLK_2 :                136C C |  SPRBLK_3 :                 139F C |
 SPRBLK_HARD :               40 - | *SPRBLK_WARN :                20 - |
 SPRBLOCK :                1353 C |  SPRB_LP :                  2475 C |
 SPRB_SEEK :               2493 C |  SPRCHK2 :                  23E7 C |
 SPRCHKSUM :               23E3 C |  SPRCHK_LP :                23EF C |
 SPRCNT_1 :                2525 C | *SPRCNT_ABORT :               11 - |
 SPRCNT_END :              2545 C |  SPRCOUNT :                 1545 C |
 SPRTBL_TYPE :                8 - | *SPRTBL_VECTOR :            1049 C |
 SPRTBL_WARN :               40 - |  SPRTHRESH :                   3 - |
```

```
 SPRWRV_1 :                 1395 C |  SPR_END :                   243C C |
 SPR_ENTER :                13EA C |  SPR_LDBUF2 :               1461 C |
 SPR_LDWRBUF :              1465 C |  SPR_NEXT :                 24D9 C |
 SPR_ODD :                  2436 C |  SPR_TO_RBUF :              207F C |
 SPR_TO_WRBUF :             20AC C |  SPR_WRVER :                1373 C |
 SRCHCACHE :                165E C |  SRCHDONE :                 14F2 C |
 SRCHLP :                   1475 C |  SRCHLPELSE :               14BE C |
 SRCHSPTABL :               146E C |  SRCH_C_END :               1697 C |
 SRCH_C_LP :                1662 C |  SRCH_MORE :                169F C |
 SRCH_M_1 :                 16A8 C |  SRCH_SP1 :                 14D4 C |
 SRCH_SPARE :               14B0 C |  SRVOCMNDBUF :              14B1 C |
 SRVORCVRY :                2954 C |  SRVO_LD_WT :               28C6 C |
*SRVO_R_ABORT :                A - |  SRVO_R_LEAVE :             2975 C |
 SRVO_R_RPT :               2959 C |  SRVO_R_SOK :               2964 C |
 SRVO_ST_1 :                2894 C |  SRVO_ST_2 :                28A2 C |
 SRVO_ST_EXIT :             28B7 C |  SRVO_ST_RPT :              288E C |
 SRVO_WT_1 :                28AE C |  SRV_C_END :                2858 C |
 SRV_C_LP :                 2846 C | *SRV_C_LP1 :                283C C |
 SRV_ST_END :               287A C |  SSTATUS0 :                 14B6 C |
 SS_NOHDR :                  5A2 - |  SS_OPFAIL :                 587 - |
 SS_READERR :                595 - |  SS_SPRWARN :                5A8 - |
 STACKPTR :                 174C C | *STACK_EXT :                  0 - |
*STACK_IN :                    4 - |  STACK_TOP :                 80 - |
 STARTBLOCK :               174E - | *STARTL :                     0 - |
 START_CHKDIAG :            11DC C |  START_COMMAND :            1195 C |
*START_MASK :                  9 - | *START_STATE :                0 - |
*START_VECTOR :              100A C |  STAT5_LP :                 1AA7 C |
 STATE_FAIL :                 4 - |  STATUSARRAY :              1015 - |
 STATUSPORT :               1F00 - | *STAT_ABORT :                 1 - |
*STAT_NO_HDR :                 4 - |  STAT_RD_ERR :                8 - |
 STAT_SEEK :                   2 - |  STAT_SLFTST :                8 - |
 STAT_SPARE :                  4 - |  STAT_SRVO :                  2 - |
 STAT_TABLE :               1A50 C | *STDDEFZ8INC :                1 - |
 STORE_TOS :                225D C | *STRT_CMND :                  0 - |
 STRT_FREEPROCESS :         2E42 C |  STRT_SWAP :                11EB C |
*STRT_ZH :                  11C5 C |  ST_3_LDEI :                2C45 C |
 ST_AO_RESPONSE :              E - |  ST_BLK :                   2C7B C |
 ST_LOAD_CMND :             11B1 C |  ST_L_LP :                  11BB C |
*ST_MAP_RESPONSE :             E - |  ST_MTRSPD :                2A38 C |
 ST_MTR_GD :                2A46 C |  ST_PBLOCK :                2E28 C |
 ST_THS :                   2C3D C |  ST_TOS :                   226F C |
 SUB3 :                      3A5 - |  SYS_CHK_FTL :              1CCE C |
 SYS_CMNDS :                   2 - |  SYS_INC_BLK :              1D2C C |
 SYS_INC_LP :               1D3D C |  SYS_INST :                 1281 C |
 SYS_LOG_OFFSET :              3 - |  SYS_RDERR :                1CAE C |
 SYS_RDERR1 :               1CB3 C |  SYS_RD_BB :                1CC7 C |
 SYS_RD_LP :                1C8A C |  SYS_RD_MORE :              1CA9 C |
 SYS_RD_N1 :                1C9E C |  SYS_RD_NEXT :              1C99 C |
*SYS_RD_OVRLP :             1C94 C |  SYS_RD_RESP :                22 - |
 SYS_RD_SEEK :              1C7A C |  SYS_READ :                 1C74 C |
 SYS_RW_SEEK :              1D1D C |  SYS_SET1 :                 1CF8 C |
 SYS_SETUP :                1CE3 C |  SYS_WRCHK :                1DA4 C |
 SYS_WRE1 :                 1DAE C |  SYS_WREND_RESP :              27 - |
 SYS_WRERR :                1D9C C |  SYS_WREX_RESP :               A3 - |
 SYS_WRITE :                1D48 C |  SYS_WRV :                  193D C |
 SYS_WRVER_RESP :              24 - |  SYS_WR_BB :                1DB3 C |
 SYS_WR_HS :                1D8E C |  SYS_WR_LP :                1D63 C |
 SYS_WR_MORE :              1D97 C |  SYS_WR_NEXT :              1D7D C |
*SYS_WR_OVRLP :             1D72 C |  SYS_WR_RESP :                23 - |
 SYS_WR_SEEK :              1D4E C |  S_A_XOR_END :              2FCC C |
 S_A_XOR_LP :               2FB3 C |  S_BLK_END :                13E7 C |
 S_BLK_NEW :                1414 C |  S_BLK_RPT :                13DA C |
 S_BLK_UNUSE :              1408 C |  S_BLOCK :                    1 - |
*S_CMND_BYTE :                 0 - |  S_CMND_LEN :                 5 - |
 S_CNT_1 :                  2ACD C |  S_CNT_3 :                  2AD5 C |
 S_CNT_END :                2AE5 C |  S_C_BADEND :               253F C |
 S_C_BADINC :               253E C |  S_C_SPARE :                2543 C |
*S_DIFF_BYTE :                 1 - |  S_GLBL_CYL :               2750 C |
 S_LOAD :                      1 - |  S_NORM_STATUS :              1 - |
*S_OFF_BYTE :                  2 - |  S_OK_END :                 2789 C |
 S_PARK_RESP :                 A - | *S_RATE_19_2 :                0 - |
 S_RATE_57_6 :                80 - |  S_RESTORE :                1B5F C |
 S_RSTR_RESPONSE :             7 - | *S_RST_1 :                  29B6 C |
 S_RST_ABORT :                 3 - | *S_RST_ABT1 :               29BE C |
 S_RST_ABT2 :               29F0 C |  S_RST_BD1 :                29C3 C |
 S_RST_COMM :               29EE C |  S_RST_LD :                 29F5 C |
 S_SCMND :                  1B0A C |  S_SEEKN_LP :               2A14 C |
 S_SEEK_RESPONSE :             6 - |  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 :                    2FDA C |
 TEST0_DONE :               2FE6 C |  TESTBITMAP :                80 - |
 TESTMOD8 :                 2FCF C | *TIME :                 16:16:01 - |
*TIMEOUT :                     1 - | *TIMER0 :                    10 - |
 TIMER1 :                     20 - |  TLBLOCK :                  1751 - |
 TMR :                        F1 - |  TOPOFSTACK :               17FF - |
*TOTEM_POL :                   1 - |  TRACKCOUNT :               2AB1 C |
*TRUE :                        1 - |  TSC_BITMAP :               17BE C |
*TSC_CLEAR :                17E9 C |  TSC_END :                  17F3 C |
 TSC_LP1 :                  17C2 C |  TSC_LP2 :                  17D9 C |
 TSC_MAP :                  17DD C |  TSC_SCEND :                17F1 C |
 TSC_SET :                  17EF C |  TSTMD8_DONE :              2FD9 C |
 TST_HEAD :                    0 - |  TST_HICYL :                  2 - |
 TST_LOCYL :                   5 - |  TST_SCTR :                   0 - |
 UD_C_C_END :               20F7 C | *UNUSED_REG :                59 - |
 UPDATE_1 :                 2452 C |  UPDATE_2 :                 246A C |
 UPDATE_CUR_CYL :           20EB C |  UPDATE_ERR :               2186 C |
 UPDATE_HDR :               2166 C | *UPDATE_H_END :             2188 C |
*UPDATE_ONTRCK :            2171 C |  UPDATE_SPRTBL :            243F C |
 UPDT_RECAL :               2192 C |  USEABLE :                   20 - |
 USED :                       40 - |  USER_TYPE :                  2 - |
 VCTRB0_RET :               1080 C | *VERSION :                  142F - |
 WAIT :                       40 - |  WAIT_FOR_RDY :             2668 C |
*WBUF1CRC :                 1235 C | *WBUF1ECC :                 1237 C |
*WBUF1PW :                  123F C |  WBUFFER1 :                 1021 C |
 WDATAGAP :                 1011 C |  WDATASYNC :                101F C |
*WENDGAP :                  123D C | *WHDRGAP :                  100A C |
 WHEADER :                  100B C |  WIDGET :                     1 - |
 WRBLKFENCE :               1270 C |  WRBLK_ABNORM :             1E41 C |
*WRBLK_ABORT :                 6 - |  WRBLK_END :                1E2E C |
```

```
 WRBLK_NOHDR :                1E49 C | *WRBLK_NORM :                1E0B C |
 WRBLK_NVLD :                 1E59 C |  WRBLK_RPT :                 1DC1 C |
 WRBLK_RPT1 :                 1E38 C |  WRBLK_SERVOERR :            1E54 C |
 WRBLK_UNTIL :                1E22 C |  WRBLK_VECTOR :              107D C |
 WRBUF_OR :                     40 - |  WRBUF_TO_BUF2 :             2065 C |
 WRBUF_TO_SPR :               2093 C |  WRERRCNT :                    29 - |
 WRERROR :                      80 - |  WRGAP_LP :                  1DCC C |
 WRITEARRAY :                 1000 C |  WRITEBLOCK :                1DB8 C |
 WRITE_FAST :                 1DDC C | *WRITE_OP :                    0 - |
*WRK_CNTRL :                    F0 - | *WRK_EXCEPT :                  20 - |
 WRK_IO :                       0 - |  WRK_SCR :                     40 - |
 WRK_SYS :                      10 - |  WRK_SYS2 :                    30 - |
 WRNOHDRFND :                   10 - |  WRSRVOERR :                   40 - |
 WRSTAT :                       28 - |  WRSUCCESS :                   20 - |
 WRTNVLDL :                     40 - |  WRVER_COMMON :              1964 C |
 WRVER_LEAVE :                1961 C |  WRVER_RESPONSE :              4 - |
 WRVER_RET :                  198C C |  WRV_CHGEX :                 198A C |
 WR_BBLOCK :                  18C2 C |  WR_CMD_END :                1933 C |
*WR_CMN_ABORT :                 E - |  WR_CMN_HDR :                192C C |
 WR_CMN_LP :                  1904 C |  WR_CMN_OK :                 18E8 C |
*WR_CMN_SERVO :               191D C |  WR_COMMON :                 18CF C |
*WR_HDRERR :                  1E4C C |  WR_LEAVE :                  1892 C |
 WR_OP :                        20 - |  WR_RESIDENT :                30C - |
 WR_RESPONSE :                   3 - | *WR_SERVOOK :                1E1A C |
 WR_SPR1 :                    1BC3 C |  WR_SPRTBL :                 1BA8 C |
 WR_SPR_RESP :                  10 - | *WSCTRGAP :                  1000 C |
 WTRDY_END :                  2685 C |  WTRDY_TMR :                 267C C |
 W_10MB :                        1 - |  W_20MB :                      0 - |
 W_40MB :                        0 - |  X_BADBLOCK :                1709 C |
 X_BB_LP :                    170B C |  X_HDRBADBLOCK :             172D C |
 X_HDRSPARE :                 1753 C |  X_HDR_CRCT :                174A C |
 X_READERR :                  171D C |  X_SPARE :                   1702 C |
 X_SPR_CALL :                 1704 C |  X_TO_BUF2 :                 20A2 C |
 X_TO_SPR :                   208D C |  X_UNDETER :                 16FD C |
 YMASK :                         F - | *Z8TESTL :                     0 - |
*Z8_APPLE :                     30 - | *Z8_MEM :                     10 - |
 ZEROBLK_LP :                 2280 C |  ZEROBLOCK :                 227A C |
 ZEROHDR_LP :                 2293 C |  ZEROHEADER :                228B C |
 ZERO_ELEMENT :               1621 C |  ZERO_E_LP :                 162A C |
 ZERO_FLP1 :                  20C9 C |  ZERO_FMT :                  20B8 C |
 ZERO_RDBUF :                 20B6 C |  ZONESHIFT :                   5 - |
 ZONE_TABLE :                 169A - | *ZRRD_VECTOR :               1031 C |
*ZRWCK :                        40 - |

   1253 symbols
    359 unused symbols
```

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

STANDARD (0 changed characters)


0.22 seconds assembly time

   9024 lines source file
      2 passes
      0 errors
      0 warnings
```