```
       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 - Z8 firmware
       8/       0 :                     ;                           Version 341-0288-A
       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 :                     ;                (Z8 code did not change since 1-A.4.2)
      12/       0 :                     ;
      13/       0 :                     ;
      14/       0 :                     ;                      disassembly by Al Kossow, 2011
      15/       0 :                     ;  commented and converted into as source by Patrick Schaefer, 2011
      16/       0 :                     ;   revised and merged with Apple's original V1-A.4.4 source, 2013
      17/       0 :                     ;
      18/       0 :                     ;    use as build 2011-08-02 or later. Last changes: 2013-07-30 PS
      19/       0 :                     ;
      20/       0 :                     ;*****************************************************************
      21/       0 :
      22/       0 : =1H                 DontListIncls   SET 1                   ; set to skip listing for .inc
      23/       0 :
      24/       0 : =1H                 W_10MB          SET 1                   ; choose your disk capacity
      25/       0 : =0H                 W_20MB          SET 0
      26/       0 : =0H                 W_40MB          SET 0
      27/       0 :
      28/       0 : =1AH                HiRevNumber     SET 01Ah                ; not used here but good to know
      29/       0 : =45H                LoRevNumber     SET 045h
      30/       0 : =800H               ROMsize         SET 2048
      31/       0 :
      32/       0 :
      33/       0 :                     ; this source contains Init.Assem (0000h), Resident.Assem (025Fh),
      34/       0 :                     ; Bs.Assem (04ABh), Int.Asssem (050Dh), Except.Assem (051Fh), and
      35/       0 :                     ; Res.Misc.Assem (05CAh)
      36/       0 :
      37/       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 :
      38/    174E :                             include vectors0289.inc         ; addresses referred to in external EPROM
(1)    1/    174E :                                     save
(1)    2/    174E : =>TRUE                      if DontListIncls
(1)   39/    174E :                                     restore
(1)   40/    174E :
      39/    174E :
      40/    174E :                     ;*****************************************************************
      41/    174E :                     ; Module Init.Assem
      42/    174E :                     ;
      43/    174E :                     ; This module contains the routines that are responsible for
      44/    174E :                     ; initializing widget
      45/    174E :                     ;
      46/    174E :                     ;       RESIDENT PROCEDURE RegTest
      47/    174E :                     ;       RESIDENT PROCEDURE StackTest
      48/    174E :                     ;       RESIDENT FUNCTION RamTest : BOOLEAN
      49/    174E :                     ;       RESIDENT FUNCTION EpromTest : BOOLEAN
      50/    174E :                     ;       START PROCEDURE : Main
      51/    174E :                     ;
      52/    174E :                     ;*****************************************************************
      53/       0 :                             org 0000h
      54/       0 : 05 0D               Int_Vec0        DW Vector0              ; illegal
      55/       2 : 05 0D               Int_Vec1        DW Vector1              ; illegal
      56/       4 : 05 0D               Int_Vec2        DW Vector2              ; illegal
      57/       6 : 05 0D               Int_Vec3        DW Vector3              ; illegal
      58/       8 : 05 0D               Int_Vec4        DW Vector4              ; illegal
      59/       A : 05 16               Int_Vec5        DW Vector5              ; T1 / Dead Man Timer
      60/       C :
      61/       C : 8F                  Start           di
      62/       D : 31 00                               srp #Wrk_Io             ; select I/O working register set
      63/       F :                             assume rp:Wrk_Io
      64/       F : B0 FE                               clr SPH                 ; initialize the stack
      65/      11 : E6 FF 80                            ld SPL, #Stack_Top
      66/      14 : E6 F8 36                            ld P01M, #P0_03_Adr+P0_47_Out+P1_Adr+Stack_In+Mem_Ext
      67/      17 : 0C 70                               ld P0, #Not_FmenL+Not_Z8TestL+Not_RdHdrH+Not_ServoRst
      68/      19 : E6 F6 46                            ld P2M, #P21_In+P22_In+P26_In
      69/      1C : 2C 99                               ld P2, #Not_StartL+Bsy+Z8_Mem+DrwL_Read
      70/      1E : E6 F7 51                            ld P3M, #B0_7_Ser+B1_6_Io+B3_4_Idm+B2_5_Io+Totem_Pol+Par_Off
      71/      21 : EC 19                               ld R14, #RamBank0 / 256 ; select RAM bank 1
      72/      23 : FC 01                               ld R15, #01
      73/      25 : 92 EE                               lde @RR14, R14
      74/      27 : EE                                  inc R14                 ; next RAM adr bit
      75/      28 : FC 00                               ld R15, #00
      76/      2A : 92 EE                               lde @RR14, R14
      77/      2C : EC 10                               ld R14, #1000h /256     ; save PwrFlg regs temporarely
      78/      2E : FC 00                               ld R15, #1000h #256
      79/      30 : DC 04                               ld R13, #4              ; move 4 registers
      80/      32 : CC 2C                               ld R12, #PwrFlg0
      81/      34 : 93 CE              Start_Loop       ldei @RR14, @R12
      82/      36 : DA FC                               djnz R13, Start_Loop
      83/      38 :
      84/      38 :
      85/      38 :                     ;*****************************************************************
      86/      38 :                     ; Diagnostic: Internal RAM / Register Test
      87/      38 :                     ;
      88/      38 :                     ;   This test is used as both a diagnostic and a selftest routine
      89/      38 :                     ;   in the Widget controller. It is intended to check the internal
      90/      38 :                     ;   functions of hte Z8 before allowing the controller to execute
      91/      38 :                     ;   any code that could be potentially dangerous to the system.
      92/      38 :                     ;
      93/      38 :                     ;   The test begins by checking working register set 0, registers 4,
      94/      38 :                     ;   5, 6, and 7, then uses those registers to test the rest of the
      95/      38 :                     ;   others.
      96/      38 :                     ;
      97/      38 :                     ; Test: All Zeros / All Ones
      98/      38 :                     ;
      99/      38 :                     ;   k:=0 {load value}
     100/      38 :                     ;   For i:=1 To 2 Do
     101/      38 :                     ;     Begin
```

```
102/      38 :                    ;       i:=120 {128 regs: -4 I/O ports, -4 regs used in test}
103/      38 :                    ;       j:=127 {highest register address}
104/      38 :                    ;       While (i>0) Do
105/      38 :                    ;         Begin
106/      38 :                    ;           RamReg[j]:=k
107/      38 :                    ;           j:=j-1
108/      38 :                    ;           i:=i-1
109/      38 :                    ;         End {While}
110/      38 :                    ;       For i:=1 To 120 Do
111/      38 :                    ;         Begin
112/      38 :                    ;           If RamReg[i]<>k Then HALT
113/      38 :                    ;           j:=j+1
114/      38 :                    ;         End
115/      38 :                    ;      k:=$FF {do all one's test next}
116/      38 :                    ;    End {For}
117/      38 :                    ;
118/      38 :                    ;*********************************************************************
119/      38 :
120/      38 : B0 E4              Start_RegTest   clr R4
121/      3A : 42 44                              or R4, R4               ; quick check
122/      3C : EB FE                              jr NZ, $                ; loop here if can't clear R4
123/      3E :                    ;
124/      3E : 58 E4                              ld R5, R4               ; bootstrap a few registers
125/      40 : 68 E4                              ld R6, R4               ;     /|\   .
126/      42 : 78 E4                              ld R7, R4               ;      |
127/      44 : 88 E4                              ld R8, R4               ;      |
128/      46 : 02 54                              add R5, R4              ;      | (all zeros)
129/      48 : 02 65                              add R6, R5              ;      |
130/      4A : 02 76                              add R7, R6              ;      |
131/      4C : 02 87                              add R8, R7              ;     \|/   .
132/      4E : EB FE                              jr NZ, $                ; loop here if can't bootstrap
133/      50 :                    ;
134/      50 : 4C FF                              ld R4, #0FFh            ; quick check
135/      52 : A6 E4 FF                           cp R4, #0FFh
136/      55 : EB FE                              jr NZ, $                ; loop here if can't set R4
137/      57 :                    ;
138/      57 : 58 E4                              ld R5, R4               ; bootstrap a few registers
139/      59 : 68 E4                              ld R6, R4               ;     /|\   .
140/      5B : 78 E4                              ld R7, R4               ;      |
141/      5D : 88 E4                              ld R8, R4               ;      |
142/      5F : 02 54                              add R5, R4              ;      | (all ones)
143/      61 : 02 65                              add R6, R5              ;      |
144/      63 : 02 76                              add R7, R6              ;      |
145/      65 : 02 87                              add R8, R7              ;     \|/   .
146/      67 : 06 E8 05                           add R8, #05             ; 5* (-1) + 5 = 0
147/      6A : EB FE                              jr NZ, $                ; loop here if can't bootstrap
148/      6C :                    ;
149/      6C :                    ; test the rest of the registers
150/      6C :                    ;
151/      6C : B0 E4                              clr R4                  ; load value
152/      6E : 5C 02                              ld R5, #RegLpTimes
153/      70 : 88 E4              RegLp1          ld R8, R4               ; remember load value
154/      72 : 6C 77                              ld R6, #RegCount-RegUsed-I_ORegUsed
155/      74 : 7C 7F                              ld R7, #HiRegAdr
156/      76 : F3 74              RegLp2          ld @R7, R4              ; begin loading RAM
157/      78 : 00 E7                              dec R7                  ; point to new reg
158/      7A : 6A FA                              djnz R6, RegLp2
159/      7C : 6C 77                              ld R6, #RegCount-RegUsed-I_ORegUsed
160/      7E : 7E                 RegLp3          inc R7
161/      7F : E3 47                              ld R4, @R7              ; read regs
162/      81 : A2 48                              cp R4, R8
163/      83 : EB FE                              jr NZ, $                ; loop here if failure
164/      85 : 6A F7                              djnz R6, RegLp3
165/      87 : 4C FF                              ld R4, #0FFh            ; set up for all ones test
166/      89 : 5A E5                              djnz R5, RegLp1
167/      8B :
168/      8B :
169/      8B :                    ;*********************************************************************
170/      8B :                    ; Diagnostic: Stack Test
171/      8B :                    ;
172/      8B :                    ;   This test is used to test hte Z8's push and pop capabilities
173/      8B :                    ;   as well as its ability to perform calls and returns.
174/      8B :                    ;
175/      8B :                    ;   The stack is set to internal, and the stack top is set to
176/      8B :                    ;   location $80. The ensuing PUSH instruction first decrements the
177/      8B :                    ;   stack pointer to $7F, and then stores the contents of the
178/      8B :                    ;   register being pushed in that location. A POP instruction
179/      8B :                    ;   should first load the register with the contents pointed
180/      8B :                    ;   to by the stack pointer and then increment the pointer.
181/      8B :                    ;
182/      8B :                    ;   Registers used: R4, R5
183/      8B :                    ;
184/      8B :                    ;*********************************************************************
185/      8B :
186/      8B : B0 FE                              clr SPH
187/      8D : E6 FF 80                           ld SPL, #80h
188/      90 : 48 FE                              ld R4, SPH
189/      92 : EB FE                              jr NZ, $                ; check the loading of Stack Pointer
190/      94 : A6 FF 80                           cp SPL, #80h
191/      97 : EB FE                              jr NZ, $
192/      99 :                    ;
193/      99 : 4C AA                              ld R4, #0AAh            ; PUSH / POP $AA
194/      9B : 70 E4                              push R4
195/      9D : 48 FE                              ld R4, SPH
196/      9F : EB FE                              jr NZ, $                ; check for decrement of pointer
197/      A1 : A6 FF 7F                           cp SPL, #7Fh
198/      A4 : EB FE                              jr NZ, $
199/      A6 : 50 E4                              pop R4
200/      A8 : A6 E4 AA                           cp R4, #0AAh
201/      AB : EB FE                              jr NZ, $
202/      AD : 48 FE                              ld R4, SPH
203/      AF : EB FE                              jr NZ, $                ; check for increment of pointer
204/      B1 : A6 FF 80                           cp SPL, #80h
205/      B4 : EB FE                              jr NZ, $
206/      B6 :                    ;
207/      B6 : D6 00 B9                           call Stk_Test           ; check call capability
208/      B9 : 48 FF              Stk_Test        ld R4, SPL
209/      BB : A6 E4 7E           Stk_Test1       cp R4, #7Eh             ; check for double dec on call
210/      BE : EB FE                              jr NZ, $
211/      C0 :                    ;
212/      C0 : 4E                                 inc R4                  ; point to low address byte
213/      C1 : E3 54                              ld R5, @R4              ; load low address byte
214/      C3 : A6 E5 B9                           cp R5, #Stk_Test        ; should be next instruction after call
215/      C6 : EB FE                              jr NZ, $
216/      C8 :                    ;
217/      C8 : E7 E4 CE                           ld @R4, #Stk_Ret        ; check return capability
218/      CB : AF                                 ret
219/      CC : 8B FE              Stk_Halt        jr Stk_Halt             ; Halt if pc just got incremented
```

```
220/      CE :                      ;
221/      CE : B0 FE                Stk_Ret         clr SPH                 ; initialize the stack
222/      D0 : E6 FF 80                             ld SPL, #Stack_Top
223/      D3 :
224/      D3 :
225/      D3 :                      ;*****************************************************************
226/      D3 :                      ;
227/      D3 :                      ; Initialize the I/O Port Control Registers
228/      D3 :                      ;
229/      D3 :                      ;   P01M -->    select Port 0, Bits 0:3 =Adr Bits 8:11
230/      D3 :                      ;               select Port 0, Bits 4:7 = Output Ports
231/      D3 :                      ;               select Port 1= Adr Bits 0:7
232/      D3 :                      ;               select Extended Memory Timing
233/      D3 :                      ;               select Internal Stack
234/      D3 :                      ;
235/      D3 :                      ;   P2M  -->    select Port 2, Bit 1 = Input
236/      D3 :                      ;               select Port 2, Bit 2 = Input
237/      D3 :                      ;               select Port 2, Bit 6 = Input
238/      D3 :                      ;               all other Port 2 bits are outputs
239/      D3 :                      ;
240/      D3 :                      ;   P3M  -->    select Port 3, Bit 0 = SIO Data In
241/      D3 :                      ;               select Port 3, Bit 1 = Tin
242/      D3 :                      ;               select Port 3, Bit 2 = Input
243/      D3 :                      ;               select Port 3, Bit 3 = Input
244/      D3 :                      ;               select Port 3, Bit 4 = Dm-
245/      D3 :                      ;               select Port 3, Bit 5 = Output
246/      D3 :                      ;               select Port 3, Bit 6 = Tout
247/      D3 :                      ;               select Port 3, Bit 7 = SIO Data Out
248/      D3 :                      ;               select SIO Parity Off
249/      D3 :                      ;               select Totem-Pole outputs for Port 2
250/      D3 :                      ;
251/      D3 :                      ;*****************************************************************
252/      D3 :
253/      D3 : 4C 16                                ld P01M_Image, #P0_03_Adr+P0_47_Out+P1_Adr+Stack_In
254/      D5 : 6C 1C                                ld P01M_StMach, #P0_03_Out+P0_47_Out+P1_Tri+Stack_In
255/      D7 : 49 F8                                ld P01M, P01M_Image
256/      D9 : E6 F6 46                             ld P2M, #P21_in+P22_In+P26_In
257/      DC : 5C 51                                ld P3M_Image, #B0_7_Ser+B1_6_Io+B3_4_Idm+B2_5_Io+Totem_Pol+Par_Off
258/      DE : 7C 41                                ld P3M_StMach, #B0_7_Ser+B1_6_Io+B3_4_Io+B2_5_Io+Totem_Pol+Par_Off
259/      E0 : 59 F7                                ld P3M, P3M_Image
260/      E2 :                      ;
261/      E2 : 8C EF                                ld Dm_Mask, #0FFh-Dm
262/      E4 : 9C FE                                ld Start_Mask, #0FFh-Not_StartL
263/      E6 :                      ;
264/      E6 : E6 F9 08                             ld IPR, #08             ; Group A := 0, A > B > C
265/      E9 : B0 FB                                clr IMR                 ; disallow interrupts
266/      EB : B0 FA                                clr IRQ                 ; clear any pending interrupts
267/      ED :                      ;
268/      ED : B0 56                                clr DiskStat
269/      EF : B0 24                                clr Excpt_Stat          ; recovery off
270/      F1 : B0 5A                                clr BlkStat
271/      F3 : B0 26                                clr RdStat
272/      F5 : B0 27                                clr RdErrCnt
273/      F7 : B0 28                                clr WrStat
274/      F9 : B0 29                                clr WrErrCnt
275/      FB : B0 5B                                clr Cache_Index
276/      FD : E6 57 90                             ld Seek_Type, #Access_Offset
277/     100 : E6 58 02                             ld Data_Type, #User_Type
278/     103 : B0 2A                                clr SeekCount
279/     105 : B0 2B                                clr SeekCount+1
280/     107 :                      ;
281/     107 : EC 10                                ld R14, #Rwi_Value /256 ; load RWI/PC cylinder value
282/     109 : FC 08                                ld R15, #Rwi_Value #256
283/     10B : CC 20                                ld R12, #Hi_Rwi_Reg
284/     10D : C3 CE                                ldci @R12, @RR14
285/     10F : C3 CE                                ldci @R12, @RR14
286/     111 : B0 F1                                clr TMR                 ; initialize timers
287/     113 : E6 F3 03                             ld PRE1, #3             ; mod 64, continuous run
288/     116 : E6 F2 8F                             ld T1, #143             ; interrupt every 10ms
289/     119 : 46 F1 0C                             or TMR, #T1_CntEn+T1_Load
290/     11C : 9F                                   ei                      ; kludge for Z8 to do polling
291/     11D : 8F                                   di
292/     11E : E6 FB 20                             ld IMR, #Timer1         ; allow only Timer 1 interrupts
293/     121 :
294/     121 :
295/     121 :                      ;*****************************************************************
296/     121 :                      ;
297/     121 :                      ; Initial Port Assignments
298/     121 :                      ;
299/     121 :                      ;   Port 0 -->  Bits 0:3 are set to Adr 8:11 via P01M
300/     121 :                      ;               Bit 4 = Servo Reset active
301/     121 :                      ;               Bit 5 = Format Enable inactive
302/     121 :                      ;               Bit 6 = Z8 Test inactive
303/     121 :                      ;               Bit 7 = Read Header inactive
304/     121 :                      ;
305/     121 :                      ;   Port 1 -->  Bits 0:7 are set to Adr 0:7 via P01M
306/     121 :                      ;
307/     121 :                      ;   Port 2 -->  Bit 0 = Start inactive
308/     121 :                      ;               Bit 1 = ECC Error inactive
309/     121 :                      ;               Bit 2 = don't care / input
310/     121 :                      ;               Bit 3 = Busy active
311/     121 :                      ;               Bit 4:5 = Msel0,1: Z8 <--> Mem
312/     121 :                      ;               Bit 6 = don't care / input
313/     121 :                      ;               Bit 7 = Disk Read/Write: Read
314/     121 :                      ;
315/     121 :                      ;   Port 3  --> don't care
316/     121 :                      ;
317/     121 :                      ;*****************************************************************
318/     121 :
319/     121 : 0C 70                                ld P0, #Not_ServoRst+Not_FmenL+Not_Z8TestL+Not_RdHdrH
320/     123 : 2C 99                                ld P2, #Not_StartL+Bsy+Z8_Mem+DrwL_Read ; -67h
321/     125 :                      ;
322/     125 : 31 10                                srp #Wrk_Sys            ; context switch
323/     127 :                             assume RP:Wrk_Sys
324/     127 :                      ;
325/     127 : D6 02 00                             call Clr_BankSwitch
326/     12A :                      ;
327/     12A : 46 24 10                             or Excpt_Stat, #PwrRst  ; assume a power reset
328/     12D : 2C 10                                ld R2, #1000h /256      ; check saved power flags
329/     12F : 3C 00                                ld R3, #1000h #256
330/     131 : EC 10                                ld R14, #PassWord /256
331/     133 : FC 03                                ld R15, #PassWord #256
332/     135 : 4C 04                                ld R4, #4               ; check 4 bytes
333/     137 :                      ;
334/     137 : 82 02               PwrRst_Lp        lde R0, @RR2            ; get saved byte
335/     139 : C2 1E                                ldc R1, @RR14           ; get a byte of password
336/     13B : A0 E2                                incw RR2
337/     13D : A0 EE                                incw RR14
```

```
338/     13F : A2 01                              cp R0, R1
339/     141 : EB 05                              jr NZ, Power_On
340/     143 : 4A F2                              djnz R4, PwrRst_Lp
341/     145 :                          ;
342/     145 : 56 24 EF                           and Excpt_Stat, #0FFh-PwrRst
343/     148 :                          ;
344/     148 : E6 38 07                 Power_On  ld Scr_Cntr, #2000 /256
345/     14B : E6 39 D0                           ld Scr_Cntr+1, #2000 #256
346/     14E : E6 2C F0                           ld PwrFlg0, #0F0h       ; initialize power-on flags
347/     151 : E6 2D 78                           ld PwrFlg1, #078h
348/     154 : E6 2E 3C                           ld PwrFlg2, #03Ch
349/     157 : E6 2F 1E                           ld PwrFlg3, #01Eh
350/     15A : 8D 02 19                           jp Main                 ; go to main routine
351/     15D :
352/     15D :
353/     15D :                          ;*****************************************************************
354/     15D :                          ;
355/     15D :                          ; Function: External Ram Test
356/     15D :                          ;
357/     15D :                          ;   This test is used as both a diagnostic and a selftest routine
358/     15D :                          ;   in the Widget controller. It is intended to check the external RAM
359/     15D :                          ;   of the controller before allowing the controller to execute
360/     15D :                          ;   any code that could be potentially dangerous to the system.
361/     15D :                          ;
362/     15D :                          ; Inputs: none
363/     15D :                          ;
364/     15D :                          ; Outputs: EcternalRamTest : BOOLEAN {zero flag false if error}
365/     15D :                          ;
366/     15D :                          ; Test: All Zeros / All Ones
367/     15D :                          ;
368/     15D :                          ;  k:=$FF  {load value}
369/     15D :                          ;  For j := 1 To 2 Do
370/     15D :                          ;    Begin
371/     15D :                          ;      Dec_Scr_Ctnr
372/     15D :                          ;      i:=RAMSize
373/     15D :                          ;      RamPtr:=RamSize-1
374/     15D :                          ;      While i<>0 Do
375/     15D :                          ;        Begin
376/     15D :                          ;          Ram[RamPtr]:=k
377/     15D :                          ;          RamPtr:=RamPtr-1
378/     15D :                          ;          i:=i-1
379/     15D :                          ;        End
380/     15D :                          ;        i:=RamSize
381/     15D :                          ;        While i<>0 Do
382/     15D :                          ;          Begin
383/     15D :                          ;            If (Ram[RamPtr]<>k) Then HALT
384/     15D :                          ;            RamPtr:=RamPtr+1
385/     15D :                          ;            i:=i-1
386/     15D :                          ;          End
387/     15D :                          ;        k:=00 {do all zero's test next}
388/     15D :                          ;    End
389/     15D :                          ;
390/     15D :                          ;  RamSize = 2048
391/     15D :                          ;  k = R5
392/     15D :                          ;  j = R8
393/     15D :                          ;  i = R6, R7
394/     15D :                          ;  RamPtr = R10, R11
395/     15D :                          ;  scratch reg = R4
396/     15D :                          ;
397/     15D :                          ;*****************************************************************
398/     15D :
399/     15D : 4C FF                    RamTest   ld R4, #0FFh            ; load value
400/     15F : 8C 02                              ld R8, #RamLpTimes
401/     161 :                          ;
402/     161 : D6 01 EB                 RamLp1    call Dec_Scr_Cnt
403/     164 : 58 E4                              ld R5, R4               ; remember load value
404/     166 : 7C 00                              ld R7, #RamSize #256
405/     168 : 6C 08                              ld R6, #RamSize /256
406/     16A : BC FF                              ld R11, #HiRamAdr #256  ; Initialize RAM ptr
407/     16C : AC 17                              ld R10, #(HiRamAdr+RamOffset) /256
408/     16E : 92 4A                    RamLp2    lde @RR10, R4           ; begin loading RAM
409/     170 : 80 EA                              decw RR10               ; point to next RAM location
410/     172 : 7A FA                              djnz R7, RamLp2
411/     174 : 6A F8                              djnz R6, RamLp2
412/     176 : 6C 08                              ld R6, #RamSize /256
413/     178 :                          ;
414/     178 : A0 EA                    RamLp3    incw RR10               ; point to next RAM location to check
415/     17A : 82 4A                              lde R4, @RR10           ; read regs
416/     17C : A2 45                              cp R4, R5
417/     17E : EB 08                              jr NZ, RamTestExit      ; exit here if failure
418/     180 : 7A F6                              djnz R7, RamLp3
419/     182 : 6A F4                              djnz R6, RamLp3
420/     184 :                          ;
421/     184 : 4C 00                              ld R4, #0               ; set up for all zero test
422/     186 : 8A D9                              djnz R8, RamLp1
423/     188 : AF                       RamTestExit  ret
424/     189 :
425/     189 :
426/     189 :                          ;*****************************************************************
427/     189 :                          ;
428/     189 :                          ; Function: EPROM Test
429/     189 :                          ;
430/     189 :                          ;  This test is used as both as diagnostic for the external EPROM
431/     189 :                          ;  for the Widget Controller and as a selftest routine in the same
432/     189 :                          ;  unit's system firmware.
433/     189 :                          ;
434/     189 :                          ;  The first location in the external EPROM holds one byte that
435/     189 :                          ;  is the check byte for that EPROM. The check byte is calculated by
436/     189 :                          ;  adding then shifting each byte in the EPROM:
437/     189 :                          ;
438/     189 :                          ; Inputs: TopBank: BYTE {R0}
439/     189 :                          ;
440/     189 :                          ; Outputs: EpromTest: BOOLEAN {zero flag false if error}
441/     189 :                          ;
442/     189 :                          ; Algorithm:
443/     189 :                          ;
444/     189 :                          ;  Begin
445/     189 :                          ;   For both halves of the EPROM Do
446/     189 :                          ;    Sum:=0
447/     189 :                          ;    Sum:=Sum+256*Eprom[0]
448/     189 :                          ;    For i:=1 To MaxEpromAdress Do
449/     189 :                          ;     Dec_Scr_Cnt
450/     189 :                          ;     Sum:=Sum+Eprom[i]
451/     189 :                          ;     If (Sum=0)
452/     189 :                          ;      Then EpromTest:=True
453/     189 :                          ;      Else EpromTest:=False
454/     189 :                          ;
455/     189 :                          ; {note that the check byte is stored in the first two locations
```

```
456/     189 :                      ;  of the EPROM}
457/     189 :                      ;
458/     189 :                      ;*********************************************************************
459/     189 :
460/     189 : 48 E0                EpromTest       ld R4, R0               ; make a loop counter out of highaddress
461/     18B :                      ;
462/     18B : 08 E4                E_Lp            ld R0, R4               ; get bank to test
463/     18D : F0 E0                                swap R0                 ; turn loop count back into highaddress
464/     18F : D6 04 BE                             call LookUp_Rom         ; select that bank
465/     192 : B0 E6                                clr R6                  ; sum := 0
466/     194 : B0 E7                                clr R7
467/     196 : 8C 10                                ld R8, #EpromSize /256
468/     198 : 9C 00                                ld R9, #EpromSize #256
469/     19A : CC 10                                ld R12, #EpromOffset /256
470/     19C : DC 00                                ld R13, #EpromOffset #256
471/     19E : C2 0C                                ldc R0, @RR12           ; get Eprom[0]
472/     1A0 : 60 E0                                com R0
473/     1A2 : 02 60                                add R6, R0              ; sum := sum + 256 * Eprom[0]
474/     1A4 : A0 EC                                incw RR12               ; point to Eprom[1]
475/     1A6 : 00 E9                                dec R9                  ; account for Eprom[0]
476/     1A8 : C2 0C                                ldc R0, @RR12           ; get Eprom[1]
477/     1AA : 60 E0                                com R0
478/     1AC : 02 70                                add R7, R0              ; sum := Sum + Eprom[1]
479/     1AE : A0 E6                                incw RR6                ; dp two's complement arithmetic
480/     1B0 : A0 EC                                incw RR12               ; point to Eprom[2]
481/     1B2 : 00 E9                                dec R9                  ; account for Eprom[1]
482/     1B4 :                      ;
483/     1B4 : D6 01 EB             Eprom_Lp        call Dec_Scr_Cnt
484/     1B7 : C2 0C                                ldc R0, @RR12           ; get Eprom[i]
485/     1B9 : A0 EC                                incw RR12
486/     1BB : 02 70                                add R7, R0              ; sum := sum + Eprom[i]
487/     1BD : 16 E6 00                             adc R6, #0
488/     1C0 : 9A F2                                djnz R9, Eprom_Lp
489/     1C2 : 8A F0                                djnz R8, Eprom_Lp
490/     1C4 : EB 02                                jr NZ, Eprom_End
491/     1C6 :                      ;
492/     1C6 : 4A C3                                djnz R4, E_Lp
493/     1C8 :
494/     1C8 : 8D 04 F8             Eprom_End       jp Bank_Ret
495/     1CB :
496/     1CB :
497/     1CB :                      ;*********************************************************************
498/     1CB :                      ;
499/     1CB :                      ; Procedure: MsWait
500/     1CB :                      ;
501/     1CB :                      ;   This procedure is used as a software timing loop, where
502/     1CB :                      ;   the busy wait length is equal to .01 * InputValue in seconds
503/     1CB :                      ;   (one unit wait = 10ms)
504/     1CB :                      ;
505/     1CB :                      ; Inputs: WaitLength: WORD (RR2)
506/     1CB :                      ;
507/     1CB :                      ; Outputs: none
508/     1CB :                      ;
509/     1CB :                      ;*********************************************************************
510/     1CB :
511/     1CB : 0C 0A                MsWait          ld R0, #10              ; change LED every 100ms
512/     1CD : 56 FA DF             MsWait_1        and IRQ, #0FFh-Timer1   ; clear any pending times
513/     1D0 : 76 FA 20             MsWait_Lp       tm IRQ, #Timer1         ; wait for timer int
514/     1D3 : 6B FB                                jr Z, MsWait_Lp
515/     1D5 : 00 E0                                dec R0
516/     1D7 : EB 0D                                jr NZ, WsWait_Dec
517/     1D9 : 70 E2                                push R2                 ; save counter
518/     1DB : 70 E3                                push R3
519/     1DD : D6 03 ED                             call Invert_Led
520/     1E0 : 50 E3                                pop R3
521/     1E2 : 50 E2                                pop R2
522/     1E4 : 0C 0A                                ld R0, #10
523/     1E6 : 80 E2                WsWait_Dec      decw RR2                ; count down a unit
524/     1E8 : EB E3                                jr NZ, MsWait_1
525/     1EA : AF                                   ret
526/     1EB :
527/     1EB :
528/     1EB : 76 FA 20             Dec_Scr_Cnt     tm IRQ, #Timer1         ; wait for timer interrupt
529/     1EE : 6B 0F                                jr Z, Dec_Scr_End
530/     1F0 : D6 03 ED                             call Invert_Led
531/     1F3 : 56 FA DF                             and IRQ, #0FFh-Timer1   ; get rid of old interrupt
532/     1F6 : 08 38                                ld R0, Scr_Cntr         ; check for already zero count
533/     1F8 : 44 39 E0                             or R0, Scr_Cntr+1
534/     1FB : 6B 02                                jr Z, Dec_Scr_End
535/     1FD : 80 38                                decw Scr_Cntr
536/     1FF : AF                   Dec_Scr_End     ret
537/     200 :
538/     200 :
539/     200 : 2C 18                Clr_BankSwitch  ld R2, #BankReg /256
540/     202 : 3C 00                                ld R3, #BankReg #256
541/     204 : 1C 07                                ld R1, #7               ; clear 7 bits
542/     206 : 92 22                Clr_B_Lp        lde @RR2, R2
543/     208 : 2E                                   inc R2
544/     209 : 1A FB                                djnz R1, Clr_B_Lp
545/     20B : 08 24                                ld R0, Excpt_Stat       ; clear all but LED state
546/     20D : 56 E0 01                             and R0, #0FFh-Led_Mask
547/     210 : D6 03 E0                             call Set_Led
548/     213 : 0C 01                                ld R0, #Ram1
549/     215 : D6 03 B3                             call Set_RamBank
550/     218 : AF                                   ret
551/     219 :
552/     219 :
553/     219 :                      ;*********************************************************************
554/     219 :                      ;
555/     219 :                      ; Start_Procedure: Main
556/     219 :                      ;
557/     219 :                      ; Inputs: none
558/     219 :                      ;
559/     219 :                      ; Outputs: none
560/     219 :                      ;
561/     219 :                      ; Algorithm:
562/     219 :                      ;
563/     219 :                      ;  Begin
564/     219 :                      ;   SlfTst_Result:={assume all tests fail}
565/     219 :                      ;   SlfTst_Result.RamTest:=RamTest
566/     219 :                      ;   SlfTst_Result.EpromTest:=EpromTest
567/     219 :                      ;   Init_ExtStack
568/     219 :                      ;   ZeroBlock(ReadArray)
569/     219 :                      ;   ClearStatus
570/     219 :                      ;   Load_PassWord(WrBlkFence)
571/     219 :                      ;   SelfTest
572/     219 :                      ;   If system passed self test
573/     219 :                      ;    Then
```

```
574/     219 :                    ;     Recovery is turned on
575/     219 :                    ;     Load_SprTbl
576/     219 :                    ;     Load_Cache
577/     219 :                    ;     If SystemCode Then Scan
578/     219 :                    ;    Else Recovery is turned off
579/     219 :                    ;   Clr_Bsy(Init_Response, Not(Cmnd_Pending))
580/     219 :                    ;  End
581/     219 :                    ;
582/     219 :                    ;*****************************************************************
583/     219 :
584/     219 : E6 25 FF           Main            ld SlfTst_Result, #0FFh ; assume all failures at first
585/     21C : D6 01 5D                           call RamTest            ; test external RAM
586/     21F : EB 03                              jr NZ, Main_Eprom
587/     221 : 56 25 7F                           and SlfTst_Result, #0FFh-Ram_Fail
588/     224 :                    ;
589/     224 : 0C 02              Main_Eprom      ld R0, #Eprom2
590/     226 : D6 01 89                           call EpromTest          ; test external eprom bank 0,1
591/     229 : EB 03                              jr NZ, Main_SelfTst
592/     22B : 56 25 BF                           and SlfTst_Result, #0FFh-Eprom_Fail
593/     22E :                    ;
594/     22E : D6 10 29           Main_SelfTst    call ExtStk_Vector      ; init external stack
595/     231 : EC 12                              ld R14, #WrBlkFence /256
596/     233 : FC 70                              ld R15, #WrBlkFence #256
597/     235 : D6 10 19                           call LdPw_Vector        ; set write buffer fence
598/     238 : D6 10 39                           call ClrStat_Vector     ; clear all status'
599/     23B : D6 10 41                           call SlfTst_Vector      ; perform selftest
600/     23E :                    ;
601/     23E : 76 25 FE           Main_LdSpr      tm SlfTst_Result, #0FFh-No_SprTbl
602/     241 : 6B 05                              jr Z, Main_Set_R
603/     243 : 56 24 7F                           and Excpt_Stat, #0FFh-Recovery  ; then recovery off
604/     246 : 8B 0C                              jr Main_Lp1
605/     248 :                    ;
606/     248 : 46 24 80           Main_Set_R      or Excpt_Stat, #Recovery        ; else it is on
607/     24B : D6 10 49                           call SprTbl_Vector
608/     24E : D6 10 51           Main_Cache      call LC_Vector
609/     251 : D6 10 63                           call IScan_Vector
610/     254 : D6 02 00           Main_Lp1        call Clr_BankSwitch
611/     257 : D6 06 72                           call Set_SeekNeeded
612/     25A : E6 0A 01                           ld Wrk_Io+10, #Init_Response
613/     25D : B0 0B                              clr Wrk_Io+11           ; Cmnd_Pending; IBsy:=false
614/     25F :
615/     25F :                    ;               \ /                     ; in-line code, must be followed by Clr_Bsy
616/     25F :
617/     25F :
618/     25F :                    ;*****************************************************************
619/     25F :                    ; Module Resident.Assem
620/     25F :                    ;
621/     25F :                    ; This module contains all the routines (besides the Z8 initialization
622/     25F :                    ; procedures) that must be resident within the Z8
623/     25F :                    ;
624/     25F :                    ;       RESIDENT PROCEDURE Bsy_Clr(MemAdr    : PTR {Wrk_Io+12:13}
625/     25F :                    ;                                  Response     : BYTE {Wrk_Io+10)
626/     25F :                    ;                                  Cmnd_Pending : BIT {Wrk_Io+11/Bit 7}
627/     25F :                    ;                                  IBsy         : BIT {Wrk_Io+11/Bit 6})
628/     25F :                    ;       RESIDENT PROCEDURE Wait_Cmd(Cmnd_Pending : BOOLEAN {R11/bit 7}
629/     25F :                    ;                                  Response : BYTE {R10}
630/     25F :                    ;                                  IBsy     : BOOLEAN {R11/bit 6}
631/     25F :                    ;                                  MemAdr   : PTR {RR12})
632/     25F :                    ;       RESIDENT PROCEDURE Get_Wr_Data(Response : BYTE {R9})
633/     25F :                    ;       RESIDENT PROCEDURE Ack_Read(Response)
634/     25F :                    ;       RESIDENT FUNCTION Wr_Resident : Status : BYTE {R0}
635/     25F :                    ;       RESIDENT FUNCTION Fmt_Resident : Status : BYTE {R0}
636/     25F :                    ;       RESIDENT FUNCTION RdHdr_Resident : Status : BYTE {R0}
637/     25F :                    ;       RESIDENT FUNCTION Rd_Resident : Status : BYTE {R0}
638/     25F :                    ;       RESIDENT SubFUNCTION Start_StMach
639/     25F :                    ;       FUNCTION Sub3(A, B : 3 BYTES {R0:2, R12:14}) : 3 BYTES {R0:2}
640/     25F :                    ;       FUNCTION Add3(A, B : 3 BYTES {R0:2, R12:14}) : 3 BYTES {R0:2}
641/     25F :                    ;       PROCEDURE Set_RamBank(Ram_Bank : BYTE {R0})
642/     25F :                    ;       PROCEDURE Set_Led(State : BIT {R0/bit 0})
643/     25F :                    ;       FUNCTION LoadStatus : BYTE {R0}
644/     25F :                    ;       PROCEDURE SetStatus(StatusByte : BYTE {R0} Value : BYTE {R1})
645/     25F :                    ;       PROCEDURE Set_Dmt(Parent1, Parent2 : BYTE {R11, R0})
646/     25F :                    ;       PROCEDURE Clr_Dmt
647/     25F :                    ;       FUNCTION FormatBlock(Parent : BYTE {R8}) : BOOLEAN Status : BYTE {R0}
648/     25F :                    ;
649/     25F :                    ;*****************************************************************
650/     25F :
651/     25F :                    ;*****************************************************************
652/     25F :                    ;
653/     25F :                    ; Procedure: Clr_Bsy
654/     25F :                    ;
655/     25F :                    ; Inputs: Response:     BYTE (Wrk_Io+10)
656/     25F :                    ;         Cmnd_Pending: BIT (Wrk_Io+11/Bit 7)
657/     25F :                    ;         IBsy:         BIT (Wrk_Io+11/Bit 6)
658/     25F :                    ;         MemAdr:       PTR (RR12)
659/     25F :                    ;
660/     25F :                    ; Outputs: none
661/     25F :                    ;
662/     25F :                    ; Begin
663/     25F :                    ;  While CMD do Begin End {busy wait on CMD}
664/     25F :                    ;  Set up memory select: Apple <--> Mem
665/     25F :                    ;  Memory_Address_Register:=MemAdr
666/     25F :                    ;  Reset BSY
667/     25F :                    ;  While not(CMD) Do Begin End {busy wait on CMD-}
668/     25F :                    ;  Jump(Start_Command) {begin a new command}
669/     25F :                    ; End
670/     25F :                    ;
671/     25F :                    ;*****************************************************************
672/     25F :
673/     25F : 31 00              Clr_Bsy         srp #Wrk_Io
674/     261 :                            assume RP:Wrk_Io
675/     261 : 2C A9                              ld P2, #Not_StartL+Disk_Mem+DrwL_Read+Bsy
676/     263 : 92 CC                              lde @RR12, R12
677/     265 : 79 F7                              ld P3M, P3m_StMach
678/     267 : 52 38                              and P3, Dm_Mask         ; set DM/IoPort to low
679/     269 : 69 F8                              ld P01M, P01m_StMach
680/     26B : 2C A9                              ld P2, #Not_StartL+Disk_Mem+DrwL_Read+Bsy       ; toggle AOE
681/     26D : 76 EB 20                           tm R11, #MultiWr        ; check for multi-write command
682/     270 : 6B 04                              jr Z, Clr_Bsy_Rd
683/     272 :                    ;
684/     272 : 2C 01                              ld P2, #Not_StartL+Apple_Mem
685/     274 : 8B 02                              jr Clr_Bsy1
686/     276 :                    ;
687/     276 : 2C 81              Clr_Bsy_Rd      ld P2, #Not_StartL+Apple_Mem+DrwL_Read
688/     278 : FC 04              Clr_Bsy1        ld R15, #Cmd            ; load mask
689/     27A : CC 10                              ld R12, #Cmnd_Ptr /256  ; MemAdr := Command Buffer
690/     27C : DC 1A                              ld R13, #Cmnd_Ptr #256
691/     27E : 72 2F              Bsy_Lp2         tm P2, R15              ; test for CMD active high
```

```
692/     280 : 6B FC                              jr Z, Bsy_Lp2
693/     282 :
694/     282 :
695/     282 :                    ;*****************************************************************
696/     282 :                    ;
697/     282 :                    ; Procedure: Wait_Cmd
698/     282 :                    ;
699/     282 :                    ; Inputs: Cmnd_Pending: BOOLEAN (R11/bit 7)
700/     282 :                    ;         Response:     BYTE (R10)
701/     282 :                    ;         IBsy:         BOOLEAN (R11/bit 6)
702/     282 :                    ;         MemAdr:       PTR (RR12)
703/     282 :                    ;
704/     282 :                    ; Outputs: none
705/     282 :                    ;
706/     282 :                    ; Algorithm:
707/     282 :                    ;
708/     282 :                    ;  Begin
709/     282 :                    ;   Set up memory select: Apple <--> Z8
710/     282 :                    ;   While not(CMD) Do Begin End {busy wait on CMD-}
711/     282 :                    ;   Write(Response) to Apple
712/     282 :                    ;   Set BSY
713/     282 :                    ;   While CMD Do Begin End {busy wait on CMD}
714/     282 :                    ;   Read(Apple_Response) from Apple
715/     282 :                    ;   If (Apple_Response=$55)
716/     282 :                    ;    Then
717/     282 :                    ;     If not(IBsy)
718/     282 :                    ;      Then
719/     282 :                    ;       Memory_Address_Register:=MemAdr
720/     282 :                    ;       reset BSY
721/     282 :                    ;       While not(CMD) Do Begin End (busy wait on CMD-)
722/     282 :                    ;     Set up memory select: Z8 <--> Mem
723/     282 :                    ;     If Cmnd_Pending
724/     282 :                    ;      Then Return
725/     282 :                    ;      Else Jump(Start_Command) {begin a new command}
726/     282 :                    ;    Else Abort(Apple_Interface_Exception, Cmnd_Pending,
727/     282 :                    ;                Response, Apple_Response)
728/     282 :                    ;  End
729/     282 :                    ;
730/     282 :                    ;*****************************************************************
731/     282 :
732/     282 : 49 F8              Wait_Cmd        ld P01M, P01m_Image     ; restore port to load adr
733/     284 : 2C A1                              ld P2, #Not_StartL+Disk_Mem+DrwL_Read
734/     286 : 92 CC                              lde @RR12, R12          ; set external memory address
735/     288 : 2C B1              Wait_Cmd1       ld P2, #Not_StartL+Z8_Apple+DrwL_Read
736/     28A : E6 F8 06                           ld P01M, #P0_03_Adr+P0_47_Out+Stack_in+P1_Out
737/     28D : 18 EA                              ld P1, R10              ; Apple gets response
738/     28F : 2C B9                              ld P2, #Not_StartL+Z8_Apple+DrwL_Read+Bsy
739/     291 : FC 04                              ld R15, #Cmd
740/     293 : EC 55                              ld R14, #Apl_Ack        ; get ready for response
741/     295 : DC 40                              ld R13, #IBsy
742/     297 : 72 2F              Cmd_Lp2         tm P2, R15              ; test for CMD inactive low
743/     299 : EB FC                              jr NZ, Cmd_Lp2
744/     29B :                    ; make Port 1 an input
745/     29B : E6 F8 0E                           ld P01M, #P0_03_Adr+P0_47_Out+Stack_In+P1_In
746/     29E : 2C A9                              ld P2, #Not_StartL+Disk_Mem+DrwL_Read+Bsy       ; strobe AOE
747/     2A0 : 2C B9                              ld P2, #Not_StartL+Z8_Apple+DrwL_Read+Bsy       ; strobe AOE
748/     2A2 : A2 1E                              cp R1, R14
749/     2A4 : EB 26                              jr NZ, Cmd_NotAck
750/     2A6 :                    ;
751/     2A6 : 72 BD              Cmd_TstBsy      tm R11, R13             ; test IBsy
752/     2A8 : 6B 09                              jr Z, No_IBsy
753/     2AA :                    ; get back to system access
754/     2AA : 49 F8              Cmd_Leave       ld P01M, P01m_Image
755/     2AC : 2C 99                              ld P2, #Not_StartL+Bsy+Z8_Mem+DrwL_Read
756/     2AE : 59 F7                              ld P3M, P3m_Image
757/     2B0 : 31 10                              srp #Wrk_Sys
758/     2B2 :                                    assume RP:Wrk_Sys
759/     2B2 : AF                                 ret                     ; resume command
760/     2B3 :                    ;
761/     2B3 :                    ; Mem <--> Host(wr)
762/     2B3 :                    ; change DM to output
763/     2B3 :                            assume RP:Wrk_Io
764/     2B3 : 79 F7              No_IBsy         ld P3M, P3m_StMach
765/     2B5 : 52 38                              and P3, Dm_Mask         ; set DM/IoPort low
766/     2B7 : 2C 01                              ld P2, #Not_StartL+Apple_Mem
767/     2B9 : FC 04                              ld R15, #Cmd
768/     2BB :                    ;
769/     2BB : 72 2F              Cmd_Lp3         tm P2, R15              ; test for CMD active high
770/     2BD : 6B FC                              jr Z, Cmd_Lp3
771/     2BF : 49 F8                              ld P01M, P01m_Image
772/     2C1 : 2C 91                              ld P2, #Not_StartL+Z8_Mem+DrwL_Read
773/     2C3 : 59 F7                              ld P3M, P3m_Image
774/     2C5 : 76 EB 80                           tm R11, #Cmnd_Pending
775/     2C8 : 6D 10 0A                           jp Z, Start_Vector
776/     2CB : AF                                 ret
777/     2CC :                    ;
778/     2CC : F8 E1              Cmd_NotAck      ld R15, P1
779/     2CE : 49 F8                              ld P01M, P01m_Image
780/     2D0 : 59 F7                              ld P3M, P3m_Image
781/     2D2 : 2C 99                              ld P2, #Not_StartL+Bsy+Z8_Mem+DrwL_Read
782/     2D4 : 31 10                              srp #Wrk_Sys
783/     2D6 :                                    assume RP:Wrk_Sys
784/     2D6 : A6 0F 69                           cp Wrk_Io+15, #Free_Proc        ; check for Freeing up bus
785/     2D9 : 6D 10 21                           jp Z, Free_Vector
786/     2DC :                    ;
787/     2DC : 70 0F              Wt_Cmd_Nak      push Wrk_Io+15
788/     2DE : D6 1E 5E                           call ClrNormStat
789/     2E1 : B0 E0                              clr R0                  ; status byte 0
790/     2E3 : 1C 80                              ld R1, #Bad_55
791/     2E5 : D6 04 03                           call SetStatus
792/     2E8 : 50 E9                              pop R9
793/     2EA : D6 05 1F                           call Abort
794/     2ED :
795/     2ED :
796/     2ED :                    ;*****************************************************************
797/     2ED :                    ;
798/     2ED :                    ; Procedure: Get_Wr_Data {get write data from host}
799/     2ED :                    ;
800/     2ED :                    ; Inputs: Response: BYTE (Wrk_Io+10)
801/     2ED :                    ;
802/     2ED :                    ; Begin
803/     2ED :                    ;  Wait_Cmd(Command_Pending, Response, NotIBsy, WBuffer1)
804/     2ED :                    ;  Wait_Cmd(Command_Pending, EndWriteResponse, IBsy, WBuffer1)
805/     2ED :                    ; End
806/     2ED :                    ;
807/     2ED :                    ;*****************************************************************
808/     2ED :
809/     2ED : 31 00              Get_Wr_Data     srp #Wrk_Io
```

```
810/     2EF :                          assume RP:Wrk_Io
811/     2EF : BC 80                           ld R11, #Cmnd_Pending   ; Cmnd_Pending:=true, IBsy:=false
812/     2F1 : CC 10                           ld R12, #WBuffer1 /256
813/     2F3 : DC 21                           ld R13, #Wbuffer1 #256
814/     2F5 : D6 02 82                        call Wait_Cmd
815/     2F8 : AC 06                           ld R10, #End_Wr_Response
816/     2FA : BC C0                           ld R11, #Cmnd_Pending+IBsy      ; Cmnd_Pending:=true
817/     2FC : D6 02 82                        call Wait_Cmd
818/     2FF : 31 10                           srp #Wrk_Sys
819/     301 :                          assume RP:Wrk_Sys
820/     301 : AF                              ret
821/     302 :
822/     302 :
823/     302 :                     ;*********************************************************************
824/     302 :                     ;
825/     302 :                     ; Procedure: Ack_Read {acknowledge read command, set BSY}
826/     302 :                     ;
827/     302 :                     ; Inputs: Response: BYTE (Wrk_Io+10)
828/     302 :                     ;
829/     302 :                     ; Outputs: none
830/     302 :                     ;
831/     302 :                     ; Begin
832/     302 :                     ;  Wait_Cmd(Command_Pending, Response, Bsy, x)
833/     302 :                     ; End
834/     302 :                     ;
835/     302 :                     ;*********************************************************************
836/     302 :
837/     302 : 31 00               Ack_Read        srp #Wrk_Io
838/     304 :                          assume RP:Wrk_Io
839/     304 : BC C0                           ld R11, #Cmnd_Pending+IBsy
840/     306 : D6 02 88                        call Wait_Cmd1
841/     309 : 31 10                           srp #Wrk_Sys
842/     30B :                          assume RP:Wrk_Sys
843/     30B : AF                              ret
844/     30C :
845/     30C :
846/     30C :                     ;*********************************************************************
847/     30C :                     ;
848/     30C :                     ; Function: Wr_Resident
849/     30C :                     ;
850/     30C :                     ;  This function exists primarily because the architecture of the
851/     30C :                     ;  Widget Controller prevents the Z8 from executing instructions
852/     30C :                     ;  from external memory while the state machine is running.
853/     30C :                     ;
854/     30C :                     ; Inputs: none
855/     30C :                     ;
856/     30C :                     ; Outputs: Status: BYTE (R0)
857/     30C :                     ;
858/     30C :                     ; Algorithm:
859/     30C :                     ;
860/     30C :                     ; Begin
861/     30C :                     ;  Msel0:1 := Disk <--> Mem
862/     30C :                     ;  Set-up external RAM address counter for Write
863/     30C :                     ;  DiskRW:=Write
864/     30C :                     ;  RdHdrH:=false
865/     30C :                     ;  If (Cylinder>RWI_Cylinder)
866/     30C :                     ;   Then
867/     30C :                     ;    PC:=true
868/     30C :                     ;    RWI:=true
869/     30C :                     ;   Else
870/     30C :                     ;    PC:=false
871/     30C :                     ;    RWI:=false
872/     30C :                     ;  StartStateMachine
873/     30C :                     ; End
874/     30C :                     ;
875/     30C :                     ;*********************************************************************
876/     30C :
877/     30C : 31 00               Wr_Resident     srp #Wrk_Io
878/     30E :                          assume RP:Wrk_Io
879/     30E : 2C 29                           ld P2, #Not_StartL+Bsy+Disk_Mem
880/     310 : E8 20                           ld R14, Hi_Rwi_Reg
881/     312 : F8 21                           ld R15, Lo_Rwi_Reg
882/     314 : 24 53 EF                        sub R15, Cylinder+1     ; check for >RWI_Cylinder
883/     317 : 34 52 EE                        sbc R14, Cylinder
884/     31A : 1B 04                           jr LT, WPC_Else
885/     31C : AC 7F                           ld R10, #7Fh            ; bits inactive
886/     31E : 8B 02                           jr WPC_End
887/     320 : AC 73               WPC_Else        ld R10, #7Fh-RWI-PC
888/     322 : 8B 28               WPC_End         jr Rd_Res2
889/     324 :
890/     324 :
891/     324 :                     ;*********************************************************************
892/     324 :                     ;
893/     324 :                     ; Function: Fmt_Resident
894/     324 :                     ;
895/     324 :                     ;  This function exists primarily because the architecture of the
896/     324 :                     ;  Widget Controller prevents the Z8 from executing instructions
897/     324 :                     ;  from external program memory while the state machine is running
898/     324 :                     ;
899/     324 :                     ; Inputs: none
900/     324 :                     ;
901/     324 :                     ; Outputs: Status: BYTE (R0)
902/     324 :                     ;
903/     324 :                     ; Algorithm:
904/     324 :                     ;
905/     324 :                     ;  Begin
906/     324 :                     ;   Msel0:1 := Disk <--> Mem
907/     324 :                     ;   Set-up external RAM address counter for Format
908/     324 :                     ;   DiskRW:=Write
909/     324 :                     ;   RdHdrH:=True
910/     324 :                     ;   If (Cylinder>RWI_Cylinder)
911/     324 :                     ;    Then
912/     324 :                     ;     PC:=true
913/     324 :                     ;     RWI:=true
914/     324 :                     ;    Else
915/     324 :                     ;     PC:=false
916/     324 :                     ;     RWI:=false
917/     324 :                     ;   StartStateMachine
918/     324 :                     ;  End
919/     324 :                     ;
920/     324 :                     ;*********************************************************************
921/     324 :
922/     324 : 31 00               Fmt_Resident    srp #Wrk_Io
923/     326 :                          assume RP:Wrk_Io
924/     326 : 2C 29                           ld P2, #Not_StartL+Bsy+Disk_Mem
925/     328 : E8 20                           ld R14, Hi_Rwi_Reg
926/     32A : F8 21                           ld R15, Lo_Rwi_Reg
927/     32C : 24 53 EF                        sub R15, Cylinder+1     ; check for >RWI_Cylinder
```

```
 928/     32F : 34 52 EE                          sbc R14, Cylinder
 929/     332 : 1B 04                             jr LT, FPC_Else
 930/     334 : AC FF                             ld R10, #0FFh           ; bits inactive
 931/     336 : 8B 02                             jr FPC_End
 932/     338 : AC F3                 FPC_Else    ld R10, #0FFh-RWI-PC
 933/     33A : EC 10                 FPC_End     ld R14, #FormatArray /256
 934/     33C : FC 20                             ld R15, #FormatArray #256
 935/     33E : 8B 10                             jr Start_StMach
 936/     340 :
 937/     340 :
 938/     340 :                       ;*********************************************************************
 939/     340 :                       ;
 940/     340 :                       ; Function: RdHdr_Resident
 941/     340 :                       ;
 942/     340 :                       ;  This function exists primarily because the architecture of the
 943/     340 :                       ;  Widget Controller prevents the Z8 from executing instructions
 944/     340 :                       ;  from external program memory while the state machine is running
 945/     340 :                       ;
 946/     340 :                       ; Inputs: none
 947/     340 :                       ;
 948/     340 :                       ; Outputs: Status: BYTE (R0)
 949/     340 :                       ;
 950/     340 :                       ; Algorithm:
 951/     340 :                       ;
 952/     340 :                       ;  Begin
 953/     340 :                       ;   Msel0:1 := Disk <--> Mem
 954/     340 :                       ;   Set-up external RAM address counter for Read
 955/     340 :                       ;   DiskRW:=Read
 956/     340 :                       ;   RdHdrH:=True
 957/     340 :                       ;   RWI:=false
 958/     340 :                       ;   PC:=false
 959/     340 :                       ;   StartStateMachine
 960/     340 :                       ;  End
 961/     340 :                       ;
 962/     340 :                       ;*********************************************************************
 963/     340 :
 964/     340 : 31 00                 RdHdr_Resident  srp #Wrk_Io
 965/     342 :                               assume RP:Wrk_Io
 966/     342 : AC FF                             ld R10, #0FFh           ; bits inactive
 967/     344 : 8B 04                             jr Rd_Res1
 968/     346 :
 969/     346 :
 970/     346 :                       ;*********************************************************************
 971/     346 :                       ;
 972/     346 :                       ; Function: Rd_Resident
 973/     346 :                       ;
 974/     346 :                       ;  This function exists primarily because the architecture of the
 975/     346 :                       ;  Widget Controller prevents the Z8 from executing instructions
 976/     346 :                       ;  from external program memory while the state machine is running
 977/     346 :                       ;
 978/     346 :                       ; Inputs: none
 979/     346 :                       ;
 980/     346 :                       ; Outputs: Status: BYTE (R0)
 981/     346 :                       ;
 982/     346 :                       ; Algorithm:
 983/     346 :                       ;
 984/     346 :                       ;  Begin
 985/     346 :                       ;   Msel0:1 := Disk <--> Mem
 986/     346 :                       ;   Set-up external RAM address counter for Read
 987/     346 :                       ;   DiskRW:=Read
 988/     346 :                       ;   RdHdrH:=false
 989/     346 :                       ;   RWI:=false
 990/     346 :                       ;   PC:=false
 991/     346 :                       ;   StartStateMachine
 992/     346 :                       ;  End
 993/     346 :                       ;
 994/     346 :                       ;*********************************************************************
 995/     346 :
 996/     346 : 31 00                 Rd_Resident     srp #Wrk_Io
 997/     348 :                               assume RP:Wrk_Io
 998/     348 : AC 7F                             ld R10, #7Fh            ; bits inactive
 999/     34A : 2C A9                 Rd_Res1         ld P2, #Not_StartL+Bsy+Disk_Mem+DrwL_Read
1000/     34C : EC 10                 Rd_Res2         ld R14, #ReadArray /256
1001/     34E : FC 00                                 ld R15, #ReadArray #256
1002/     350 :
1003/     350 :                       ;                \ /                    ; In-Line code for Speed!!
1004/     350 :
1005/     350 :
1006/     350 :                       ;*********************************************************************
1007/     350 :                       ;
1008/     350 :                       ; SubFunction: Start_StMach
1009/     350 :                       ;
1010/     350 :                       ;  This is a routine that is shared by all the resident state machine
1011/     350 :                       ;  routines and exists primarily for the purose of saving space. It's
1012/     350 :                       ;  function is to complete the set-up for the state machine and then
1013/     350 :                       ;  start it up and wait for it to finish.
1014/     350 :                       ;
1015/     350 :                       ; Begin
1016/     350 :                       ;  Z8 Port 3, Bit 4 := 0 {instead of DM it is an I/O port}
1017/     350 :                       ;  Z8 Port 1 := Input {keep Z8 from conflicting with Disk data}
1018/     350 :                       ;  SectorsRead := 2* NumberOfSectors
1019/     350 :                       ;  While not(SectorDnL) and (SectorsRead<>0) Do
1020/     350 :                       ;   Begin
1021/     350 :                       ;    If SectorMark
1022/     350 :                       ;     Then
1023/     350 :                       ;      SectorsRead:=SectorsRead-1
1024/     350 :                       ;      While SectorMark Do Begin End
1025/     350 :                       ;   End
1026/     350 :                       ;  Z8 Port 3, Bit 4 := DM
1027/     350 :                       ;  Z8 Port 1 := Adress/Data
1028/     350 :                       ;  Msel0:1 := Z8 <--> Mem
1029/     350 :                       ;  Status := StatusPort
1030/     350 :                       ;  If EccErr
1031/     350 :                       ;   Then Status.CrcError:=true
1032/     350 :                       ;  StartL:=false
1033/     350 :                       ; End
1034/     350 :                       ;
1035/     350 :                       ;*********************************************************************
1036/     350 :
1037/     350 : 92 EE                 Start_StMach    lde @RR14, R14          ; set buffer address
1038/     352 : 79 F7                                 ld P3M, P3m_StMach
1039/     354 : 52 38                                 and P3, Dm_Mask         ; set DM/IoPort low
1040/     356 : 69 F8                                 ld P01M, P01m_StMach
1041/     358 : 08 EA                                 ld P0, R10
1042/     35A :                       ;
1043/     35A : 76 E3 02              St_Res_1        tm P3, #SectorMark      ; test for sector mark
1044/     35D : EB FB                                 jr NZ, St_Res_1
1045/     35F : 52 29                                 and P2, Start_Mask      ; start state machine
```

```
1046/     361 : AC 01                           ld R10, #Not_StartL
1047/     363 : BC 02                           ld R11, #Not_EccError
1048/     365 : EC 08                           ld R14, #SectDnL        ; load mask
1049/     367 : FC 02                           ld R15, #SectorMark
1050/     369 : E6 1A 15                        ld Wrk_Sys+10, #(NbrSctrs+2)    ; timeout after 21 sectors
1051/     36C : D6 04 13                        call Set_Dmt
1052/     36F :                         ;
1053/     36F : 72 3F                   St_Res_2        tm P3, R15              ; count sector marks
1054/     371 : 6B 08                           jr Z, St_Res_3
1055/     373 : 00 1A                           dec Wrk_Sys+10
1056/     375 : 6B 0C                           jr Z, St_Res_4
1057/     377 :                         ;
1058/     377 : 72 3F                   St_Res_25       tm P3, R15              ; wait for mark to go away
1059/     379 : EB FC                           jr NZ, St_Res_25
1060/     37B :
1061/     37B : 62 3E                   St_Res_3        tcm P3, R14             ; wait for state machine to finish
1062/     37D : 6B F0                           jr Z, St_Res_2
1063/     37F : 62 3E                           tcm P3, R14             ; sample it twice
1064/     381 : 6B EC                           jr Z, St_Res_2
1065/     383 : 49 F8                   St_Res_4        ld P01M, P01m_Image
1066/     385 : 2C 98                           ld R2, #Bsy+Z8_Mem+DrwL_Read
1067/     387 : 59 F7                           ld P3M, P3m_Image
1068/     389 : CC 1F                           ld R12, #StatusPort /256
1069/     38B : DC 00                           ld R13, #StatusPort  #256
1070/     38D : 82 FC                           lde R15, @RR12
1071/     38F : 72 2B                           tm P2, R11
1072/     391 : 6B 08                           jr Z, Res_Ecc_Err
1073/     393 :                         ;
1074/     393 : 42 2A                   Res_StMach      or P2, R10              ; reset state machine
1075/     395 : 8F                              di                      ; clear the dead man timer
1076/     396 : 31 10                           srp #Wrk_Sys
1077/     398 :                                 assume RP:Wrk_Sys
1078/     398 : 08 0F                           ld R0, Wrk_Io+15        ; return StMach status
1079/     39A : AF                              ret
1080/     39B :                         ;
1081/     39B : 76 56 20                Res_Ecc_Err     tm DiskStat, #Wr_Op
1082/     39E : EB F3                           jr NZ, Res_StMach
1083/     3A0 : 56 EF BF                        and R15, #0FFh-WrtNvldL ; if ECC error
1084/     3A3 : 8B EE                           jr Res_StMach
1085/     3A5 :
1086/     3A5 :
1087/     3A5 :                         ;*********************************************************************
1088/     3A5 :                         ;
1089/     3A5 :                         ; Function: Sub3, Add3
1090/     3A5 :                         ;
1091/     3A5 :                         ; These functiond perform the following 24bit arithmetic operations:
1092/     3A5 :                         ;
1093/     3A5 :                         ;   Sub3: A <-- A - B
1094/     3A5 :                         ;   Add3: A <-- A + B
1095/     3A5 :                         ;
1096/     3A5 :                         ; Inputs: A: 3 BYTES (R0, R1, R2)
1097/     3A5 :                         ;         B: 3 BYTES (R12, R13, R14)
1098/     3A5 :                         ;
1099/     3A5 :                         ; Outputs: A: 3 BYTES (R0, R1, R2)
1100/     3A5 :                         ;
1101/     3A5 :                         ;*********************************************************************
1102/     3A5 :
1103/     3A5 : 22 2E                   Sub3            sub R2, R14
1104/     3A7 : 32 1D                           sbc R1, R13
1105/     3A9 : 32 0C                           sbc R0, R12
1106/     3AB : AF                              ret
1107/     3AC :
1108/     3AC : 02 2E                   Add3            add R2, R14
1109/     3AE : 12 1D                           adc R1, R13
1110/     3B0 : 12 0C                           adc R0, R12
1111/     3B2 : AF                              ret
1112/     3B3 :
1113/     3B3 :
1114/     3B3 :                         ;*********************************************************************
1115/     3B3 :                         ;
1116/     3B3 :                         ; Procedure: Set_RamBank
1117/     3B3 :                         ;
1118/     3B3 :                         ;  This procedure activated the 2k bank of RAM that is indicated by
1119/     3B3 :                         ;  the input parameter
1120/     3B3 :                         ;
1121/     3B3 :                         ; Inputs: Ram_Bank: BYTE (R0)
1122/     3B3 :                         ;
1123/     3B3 :                         ; Outputs: none
1124/     3B3 :                         ;
1125/     3B3 :                         ; Algorithm:
1126/     3B3 :                         ;
1127/     3B3 :                         ;  Begin
1128/     3B3 :                         ;    If (Ram_Bank>3) Then Abort
1129/     3B3 :                         ;    deselect all RAM bits
1130/     3B3 :                         ;    select the RAM bit pointed to by Ram_Bank
1131/     3B3 :                         ;  End
1132/     3B3 :                         ;
1133/     3B3 :                         ;*********************************************************************
1134/     3B3 :
1135/     3B3 : A6 E0 04                Set_RamBank     cp R0, #4
1136/     3B6 : 1B 03                           jr LT, Set_RB_Start
1137/     3B8 : D6 05 1F                        call Abort
1138/     3BB : 2C 03                   Set_RB_Start    ld R2, #Ram_Table /256
1139/     3BD : 3C D8                           ld R3, #Ram_Table #256
1140/     3BF : 90 E0                           rl R0                   ; multiply index by 2
1141/     3C1 : 02 30                           add R3, R0              ; index into table
1142/     3C3 : 16 E2 00                        adc R2, #0
1143/     3C6 : C2 12                           ldc R1, @RR2
1144/     3C8 : A0 E2                           incw RR2
1145/     3CA : C2 02                           ldc R0, @RR2
1146/     3CC : 2C 19                           ld R2, #RamBank0 /256   ; set the adr bits
1147/     3CE : 38 E0                           ld R3, R0
1148/     3D0 : 92 32                           lde @RR2, R3
1149/     3D2 : 2E                              inc R2
1150/     3D3 : 38 E1                           ld R3, R1
1151/     3D5 : 92 32                           lde @RR2, R3
1152/     3D7 : AF                              ret
1153/     3D8 :
1154/     3D8 : 00 00                   Ram_Table       DB 0, 0                 ; adr 13 := 0, adr 12 := 0
1155/     3DA : 00 01                           DB 0, 1                 ; adr 13 := 0, adr 12 := 1
1156/     3DC : 01 00                           DB 1, 0                 ; adr 13 := 1, adr 12 := 0
1157/     3DE : 01 01                           DB 1, 1                 ; adr 13 := 1, adr 12 := 1
1158/     3E0 :
1159/     3E0 :
1160/     3E0 :                         ;*********************************************************************
1161/     3E0 :                         ;
1162/     3E0 :                         ; Procedure: Set_Led
1163/     3E0 :                         ;
```

```
1164/     3E0 :                     ;  This procedure changes the state of the controller LED. The
1165/     3E0 :                     ;  state is determined by the value of State (if State is odd
1166/     3E0 :                     ;  then the Led is lit, otherwise it is turned off).
1167/     3E0 :                     ;
1168/     3E0 :                     ; Inputs: State: BIT (R0/Bit 0)
1169/     3E0 :                     ;
1170/     3E0 :                     ; Outputs: none
1171/     3E0 :                     ;
1172/     3E0 :                     ; Algorithm:
1173/     3E0 :                     ;
1174/     3E0 :                     ;  Begin
1175/     3E0 :                     ;   Led[State]:=State
1176/     3E0 :                     ;   Bank_Image.Led:=State
1177/     3E0 :                     ;  End
1178/     3E0 :                     ;
1179/     3E0 :                     ;*********************************************************************
1180/     3E0 :
1181/     3E0 : 56 24 FE            Set_Led         and Excpt_Stat, #0FFh-LedStat
1182/     3E3 : 44 E0 24                            or Excpt_Stat, R0
1183/     3E6 : 2C 18                               ld R2, #Led / 256
1184/     3E8 : 38 E0                               ld R3, R0
1185/     3EA : 92 32                               lde @RR2, R3
1186/     3EC : AF                                  ret
1187/     3ED :                     ;
1188/     3ED : 08 24               Invert_Led      ld R0, Excpt_Stat
1189/     3EF : B6 E0 01                            xor R0, #0FFh-Led_Mask  ; invert only the Led bit
1190/     3F2 : 8B EC                               jr Set_Led
1191/     3F4 :                     ;
1192/     3F4 : 2C 00               Led_Wait        ld R2, #0               ; 500 ms
1193/     3F6 : 3C 32                               ld R3, #50
1194/     3F8 : D6 01 CB                            call MsWait
1195/     3FB : AF                                  ret
1196/     3FC :
1197/     3FC :
1198/     3FC :                     ;*********************************************************************
1199/     3FC :                     ;
1200/     3FC :                     ; Function: LoadStatus
1201/     3FC :                     ;
1202/     3FC :                     ;
1203/     3FC :                     ; Inputs: none
1204/     3FC :                     ;
1205/     3FC :                     ; Outputs: LoadStatus: BYTE (R0)
1206/     3FC :                     ;
1207/     3FC :                     ;*********************************************************************
1208/     3FC :
1209/     3FC : 2C 1F               LoadStatus      ld R2, #StatusPort /256
1210/     3FE : 3C 00                               ld R3, #0
1211/     400 : 82 02                               lde R0, @RR2
1212/     402 : AF                                  ret
1213/     403 :
1214/     403 :
1215/     403 :                     ;*********************************************************************
1216/     403 :                     ;
1217/     403 :                     ; Procedure: SetStatus
1218/     403 :                     ;
1219/     403 :                     ;  SetStatus is used to set a particular bit or bits
1220/     403 :                     ;  within a specific byte of standard status
1221/     403 :                     ;
1222/     403 :                     ; Inputs: StatusByte: BYTE (R0)
1223/     403 :                     ;
1224/     403 :                     ; Outputs: none
1225/     403 :                     ;
1226/     403 :                     ;*********************************************************************
1227/     403 :
1228/     403 : 2C 14               SetStatus       ld R2, #CStatus0 /256
1229/     405 : 3C 95                               ld R3, #CStatus0 #256
1230/     407 : 02 30                               add R3, R0
1231/     409 : 16 E2 00                            adc R2, #0
1232/     40C : 82 02                               lde R0, @RR2
1233/     40E : 42 01                               or R0, R1
1234/     410 : 92 02                               lde @RR2, R0
1235/     412 : AF                                  ret
1236/     413 :
1237/     413 :
1238/     413 :                     ;*********************************************************************
1239/     413 :                     ;
1240/     413 :                     ; Procedure: Set_Dmt (dead man timer = watchdog)
1241/     413 :                     ;
1242/     413 :                     ;  This procedure sets the DeadManTimer bit, thus enabling the
1243/     413 :                     ;  decrementing of the DeadManCounter every time a timer interrupt
1244/     413 :                     ;  occurs. The Set_Dmt routine also initializes the DeadManCounter as
1245/     413 :                     ;  well as storing away information concerning the process(s) that
1246/     413 :                     ;  started the Dmt.
1247/     413 :                     ;
1248/     413 :                     ; Inputs: none
1249/     413 :                     ;
1250/     413 :                     ; Outputs: none
1251/     413 :                     ;
1252/     413 :                     ; Algorithm:
1253/     413 :                     ;
1254/     413 :                     ;  Begin
1255/     413 :                     ;   Disable Interrupts
1256/     413 :                     ;   DeadManCounter:=5s (assume timer interrupts every 10ms)
1257/     413 :                     ;   Enable Interrupts
1258/     413 :                     ;  End
1259/     413 :                     ;
1260/     413 :                     ;*********************************************************************
1261/     413 :
1262/     413 : 8F                  Set_Dmt         di
1263/     414 : E6 36 01                            ld Dmt_Counter, #Dmt_Val /256
1264/     417 : E6 37 F4                            ld Dmt_Counter+1, #Dmt_Val #256
1265/     41A : 56 FA DF                            and IRQ, #0FFh-Timer1   ; clear old events
1266/     41D : 9F                                  ei
1267/     41E : AF                                  ret
1268/     41F :
1269/     41F :
1270/     41F :                     ;*********************************************************************
1271/     41F :                     ;
1272/     41F :                     ; Procedure: Clr_Dmt (dead man timer = watchdog)
1273/     41F :                     ;
1274/     41F :                     ;  This procedure clears the DeadManTimer bit, thus disabling the
1275/     41F :                     ;  decrementing of the DeadManTimerCounter. This routine also enables
1276/     41F :                     ;  the NormalTimer.
1277/     41F :                     ;
1278/     41F :                     ; Inputs: none
1279/     41F :                     ;
1280/     41F :                     ; Outputs: none
1281/     41F :                     ;
```

```
1282/     41F :                     ;*********************************************************************
1283/     41F :
1284/     41F : 8F                  Clr_Dmt         di
1285/     420 : AF                                  ret
1286/     421 :
1287/     421 :
1288/     421 :                     ;*********************************************************************
1289/     421 :                     ;
1290/     421 :                     ; Procedure Fragment: Chk_Park1
1291/     421 :                     ;
1292/     421 :                     ;  This procedure performs a busy wait for RR2 x 10ms while
1293/     421 :                     ;  monitoring the CMD line. In case of a host request, Wait_Cmd
1294/     421 :                     ;  is entered.
1295/     421 :                     ;  This part of Strt_FreeProcess was moved to resident program space
1296/     421 :                     ;  mainly for debugging purposes. I wanted a place in the code where
1297/     421 :                     ;  the DM could be halted AFTER the host had released the controller
1298/     421 :                     ;  as a process.
1299/     421 :                     ;
1300/     421 :                     ; Inputs: WaitTime: WORD {RR2}
1301/     421 :                     ;
1302/     421 :                     ; Outputs: none
1303/     421 :                     ;
1304/     421 :                     ;*********************************************************************
1305/     421 :
1306/     421 : 56 02 F7            Chk_Park1       and P2, #0FFh-Bsy       ; allow the host to set CMD
1307/     424 : 56 FA DF            Chk_Pk3         and IRQ, #0FFh-Timer1   ; clear any old interrupts
1308/     427 : 76 02 04            Chk_Park2       tm P2, #Cmd             ; check for host wanting our attention
1309/     42A : EB 0A                               jr NZ, FreeP_Leave
1310/     42C : 76 FA 20                            tm IRQ, #Timer1         ; nothing to do, wait for 10ms timer int
1311/     42F : 6B F6                               jr Z, Chk_Park2
1312/     431 : 80 E2                               decw RR2
1313/     433 : EB EF                               jr NZ, Chk_Pk3
1314/     435 : AF                                  ret
1315/     436 :                     ;
1316/     436 : D6 02 00            FreeP_Leave     call Clr_BankSwitch     ; get to a known state
1317/     439 : 31 00                               srp #Wrk_Io
1318/     43B :                             assume RP:Wrk_Io
1319/     43B : AC 01                               ld R10, #1              ; initial response on bus
1320/     43D : BC 00                               ld R11, #0              ; Cmnd_Pending, IBsy := False
1321/     43F : CC 10                               ld R12, #Cmnd_Ptr /256  ; store command bytes here
1322/     441 : DC 1A                               ld R13, #Cmnd_Ptr #256
1323/     443 : 8D 02 82                            jp Wait_Cmd
1324/     446 :
1325/     446 :
1326/     446 :                     ;*********************************************************************
1327/     446 :                     ;
1328/     446 :                     ; Procedure: FormatBlock
1329/     446 :                     ;
1330/     446 :                     ;  This function performs the actual formatting of a sector. It is
1331/     446 :                     ;  assumed that the heads are positioned over the correct cylinder
1332/     446 :                     ;  and that the correct head has been selected. The header that is
1333/     446 :                     ;  laid down on the track is derived from the information in the
1334/     446 :                     ;  global variables in cylinder, head, and sector. It is also assumed
1335/     446 :                     ;  that memory space FormatArray has been initialized to all zeros
1336/     446 :                     ;  before entering this routine.
1337/     446 :                     ;
1338/     446 :                     ; Inputs:       Parent : BYTE {R8}
1339/     446 :                     ;
1340/     446 :                     ; Outputs:      FmtBlock    : BOOLEAN {Zero flag, true if error in ReadBlock}
1341/     446 :                     ;               Status      : BYTE {R0}
1342/     446 :                     ;
1343/     446 :                     ; Global Variables Used: Cylinder, Head, Sector, Recovery
1344/     446 :                     ;
1345/     446 :                     ;Local Variables Used:  FmtError    : BOOLEAN {R9/bit 7}
1346/     446 :                     ;                       FmtExcept   : BOOLEAN {R9/bit 6}
1347/     446 :                     ;                       FmtSuccess  : BOOLEAN {R9/bit 5}
1348/     446 :                     ;
1349/     446 :                     ; Algorithm:
1350/     446 :                     ;
1351/     446 :                     ; BEGIN
1352/     446 :                     ;   SetDeadManTimer( FormatBlock, Parent )
1353/     446 :                     ;   FmtRetryCnt := 10
1354/     446 :                     ;   FmtErrCnt := 0
1355/     446 :                     ;   FmtError := False
1356/     446 :                     ;   FmtExcept := False
1357/     446 :                     ;   SectorsRead := 2 * NbrSctrs {try to find header for two rotations}
1358/     446 :                     ;   FHdrSync := $0100
1359/     446 :                     ;   FHeader[ 1 ] := HiCylinder
1360/     446 :                     ;   FHeader[ 2 ] := LoCylinder
1361/     446 :                     ;   FHeader[ 3 ]/bits 7:6 := Head
1362/     446 :                     ;   FHeader[ 3 ]/bits 5:0 := Sector
1363/     446 :                     ;   FHeader[ 4 ] := Invert( FHeader[ 1 ] )
1364/     446 :                     ;   FHeader[ 5 ] := Invert( FHeader[ 2 ] )
1365/     446 :                     ;   FHeader[ 6 ] := Invert( FHeader[ 3 ] )
1366/     446 :                     ;     _
1367/     446 :                     ;    /
1368/     446 :                     ; R  |  Set-up external ram address counter for FORMAT
1369/     446 :                     ; E  |  Msel0:1 := Disk <--> Mem
1370/     446 :                     ; S  |  WHILE SectorMark DO BEGIN END
1371/     446 :                     ; I  |  StartL := True
1372/     446 :                     ; D  |  WHILE NOT( SectorDnL ) DO BEGIN END
1373/     446 :                     ; E  |  Status := Status_Port
1374/     446 :                     ; N  |  StartL := False
1375/     446 :                     ; T  |  Msel0:1 := Z8 <--> Mem
1376/     446 :                     ;    \_
1377/     446 :                     ;
1378/     446 :                     ;   IF NOT( Status.State = NormFmtState )
1379/     446 :                     ;     THEN
1380/     446 :                     ;       Reset_StateMachine
1381/     446 :                     ;       Abort
1382/     446 :                     ;  IF Status.ServoErr OR NOT( ServoRdy )
1383/     446 :                     ;    THEN
1384/     446 :                     ;      FmtError := True
1385/     446 :                     ;      FmtExcept := True
1386/     446 :                     ;  IF Status.WtNvldL
1387/     446 :                     ;    THEN FmtError := True
1388/     446 :                     ;      ClearDeadManTimer
1389/     446 :                     ;      Status/bit 7 := FmtError
1390/     446 :                     ;      Status/bit 6 := FmtExcept
1391/     446 :                     ; END
1392/     446 :                     ;
1393/     446 :                     ;*********************************************************************
1394/     446 :
1395/     446 : B0 E9               FormatBlock     clr R9                  ; clear booleans
1396/     448 : 2C 10                               ld R2, #FmtDelay /256
1397/     44A : 3C 07                               ld R3, #FmtDelay #256
1398/     44C : C2 12                               ldc R1, @RR2            ; get delay value
1399/     44E : 2C 10                               ld R2, #FormatArray /256
```

```
1400/     450 : 3C 20                             ld R3, #FormatArray #256
1401/     452 : B0 E0                             clr R0
1402/     454 : 92 02               FmtBlk_1      lde @RR2, R0            ; initialize gaps
1403/     456 : A0 E2                             incw RR2
1404/     458 : 1A FA                             djnz R1, FmtBlk_1
1405/     45A : 2C 10                             ld R2, #FHdrSync /256
1406/     45C : 3C 3A                             ld R3, #FHdrSync #256
1407/     45E : 0C 01                             ld R0, #1               ; load header sync
1408/     460 : 92 02                             lde @RR2, R0
1409/     462 : A0 E2                             incw RR2
1410/     464 : B0 E0                             clr R0
1411/     466 : 92 02                             lde @RR2, R0
1412/     468 : A0 E2                             incw RR2
1413/     46A : D6 10 10                          call LH_Vector
1414/     46D : 2C 10                             ld R2, #FDataSync /256  ; load data sync bit
1415/     46F : 3C 50                             ld R3, #FDataSync #256
1416/     471 : 0C 01                             ld R0, #1
1417/     473 : 92 02                             lde @RR2, R0
1418/     475 : D6 03 24                          call Fmt_Resident       ; go internal to the Z8
1419/     478 : 18 E0                             ld R1, R0               ; CASE Status.State
1420/     47A : 56 E1 0F                          and R1, #00Fh
1421/     47D : A6 E1 0A                          cp R1, #NormFmt_State
1422/     480 : 6B 08                             jr Z, Fmt_Norm
1423/     482 : D6 05 AE                          call Reset_StMach
1424/     485 : A8 E0                             ld R10, R0
1425/     487 : D6 05 1F                          call Abort
1426/     48A :                     ;
1427/     48A : 18 E0               Fmt_Norm      ld R1, R0               ; IF ServorErr OR NOT( ServoRdy )
1428/     48C : 76 E1 10                          tm R1, #ServoErr        ; servo ok?
1429/     48F : EB 05                             jr NZ, Fmt_ServoErr
1430/     491 : 76 E1 20                          tm R1, #ServoRdy
1431/     494 : EB 05                             jr NZ, Fmt_SrvoOk
1432/     496 :                     ;
1433/     496 : 46 E9 C0            Fmt_ServoErr  or R9, #FmtError+FmtSrvoErr     ; THEN FmtError AND FmtSrvoErr
1434/     499 : 8B 08                             jr FmtBlk_End
1435/     49B :                     ;
1436/     49B : 76 E0 40            Fmt_SrvoOk    tm R0, #WrtNvldL        ; IF Status.WrtNvldL (ECC error)
1437/     49E : EB 03                             jr NZ, FmtBlk_End
1438/     4A0 : 46 E9 80                          or R9, #FmtError
1439/     4A3 : 08 E9               FmtBlk_End    ld R0, R9               ; send status back to caller
1440/     4A5 : 99 28                             ld WrStat, R9
1441/     4A7 : 66 E0 80                          tcm R0, #FmtError       ; set zero flag if error
1442/     4AA : AF                                ret
1443/     4AB :
1444/     4AB :
1445/     4AB :
1446/     4AB :                     ;*********************************************************************
1447/     4AB :                     ; Module BS.Assem
1448/     4AB :                     ;
1449/     4AB :                     ; This module contains the routines that are used in bankswitching
1450/     4AB :                     ;
1451/     4AB :                     ;       PROCEDURE Bank_Call(Address : PTR {RR2})
1452/     4AB :                     ;       PROCEDURE LookUp_Rom(HiAddress : BYTE {R0})
1453/     4AB :                     ;       PROCEDURE Bank_Ret
1454/     4AB :                     ;
1455/     4AB :                     ;*********************************************************************
1456/     4AB :
1457/     4AB :                     ;*********************************************************************
1458/     4AB :                     ;
1459/     4AB :                     ; Procedure BankCall
1460/     4AB :                     ;
1461/     4AB :                     ;  This procedure manages subroutine calls from one bank of the
1462/     4AB :                     ;  controllers externel program space to the other. If one routine
1463/     4AB :                     ;  wants to call another routine in a different memory bank it must
1464/     4AB :                     ;  do indirectly through this routine.
1465/     4AB :                     ;
1466/     4AB :                     ; Inputs: Address: PTR (RR2)
1467/     4AB :                     ;
1468/     4AB :                     ; Outputs: none
1469/     4AB :                     ;
1470/     4AB :                     ; Algorithm:
1471/     4AB :                     ;
1472/     4AB :                     ;  Begin
1473/     4AB :                     ;   LookUp_Rom(Address div 256)
1474/     4AB :                     ;   Goto @Address
1475/     4AB :                     ;  End
1476/     4AB :                     ;
1477/     4AB :                     ;*********************************************************************
1478/     4AB :
1479/     4AB : 70 FD               Bank_Call       push RP
1480/     4AD : E4 FD 34                            ld Wrk_Sys2+4, RP       ; save RP for reference later
1481/     4B0 : 31 30                               srp #Wrk_Sys2
1482/     4B2 :                             assume RP:Wrk_Sys2
1483/     4B2 : 06 E4 02                            add R4, #2              ; get reg 2 in original RP
1484/     4B5 : E3 04                               ld R0, @R4              ; get hibyte of called adr
1485/     4B7 : D6 04 BE                            call LookUp_Rom
1486/     4BA : 50 FD                               pop RP
1487/     4BC : 30 E2                               jp @RR2
1488/     4BE :
1489/     4BE :
1490/     4BE :                     ;*********************************************************************
1491/     4BE :                     ;
1492/     4BE :                     ; Procedure: LookUp_Rom
1493/     4BE :                     ;
1494/     4BE :                     ;  This procedure is used to determine which of five 4k
1495/     4BE :                     ;  banks of program space the user wishes to execute within.
1496/     4BE :                     ;
1497/     4BE :                     ; Inputs: HighAddress: BYTE (R0)
1498/     4BE :                     ;
1499/     4BE :                     ; Outputs: none
1500/     4BE :                     ;
1501/     4BE :                     ; Algorithm:
1502/     4BE :                     ;
1503/     4BE :                     ;  Begin
1504/     4BE :                     ;   Case HighAddress Of
1505/     4BE :                     ;    0 : Adr13:=0, Adr12:=0
1506/     4BE :                     ;    1 : Adr13:=0, Adr12:=0
1507/     4BE :                     ;    2 : Adr13:=0, Adr12:=1
1508/     4BE :                     ;    3 : Adr13:=1, Adr12:=0
1509/     4BE :                     ;    4 : Adr13:=1, Adr12:=1
1510/     4BE :                     ;   Otherwise Abort
1511/     4BE :                     ;  End
1512/     4BE :                     ;
1513/     4BE :                     ;*********************************************************************
1514/     4BE :
1515/     4BE : 76 E0 80            LookUp_Rom      tm R0, #80h             ; check for adr out of range
1516/     4C1 : EB 08                               jr NZ, LU_Abort
1517/     4C3 : 56 E0 70                            and R0, #70h            ; mask off unnecessary stuff from address
```

```
1518/     4C6 : A6 E0 40                          cp R0, #40h
1519/     4C9 : 2B 03                             jr LE, BC_1
1520/     4CB : D6 05 1F            LU_Abort      call Abort
1521/     4CE :                     ;
1522/     4CE : 2C 04               BC_1          ld R2, #RomTable /256
1523/     4D0 : 3C EE                             ld R3, #RomTable #256
1524/     4D2 : F0 E0                             swap R0                 ; turn highaddress into index value
1525/     4D4 : 90 E0                             rl R0                   ; multiply index by 2 (2 byte/element table)
1526/     4D6 : 02 30                             add R3, R0              ; index into table
1527/     4D8 : 16 E2 00                          adc R2, #0
1528/     4DB : C2 12                             ldc R1, @RR2            ; get ROM address values
1529/     4DD : A0 E2                             incw RR2
1530/     4DF : C2 02                             ldc R0, @RR2
1531/     4E1 : 2C 1E                             ld R2, #RomBank0 /256
1532/     4E3 : 38 E0                             ld R3, R0
1533/     4E5 : 92 32                             lde @RR2, R3            ; set EpromBank0, 1
1534/     4E7 : 2C 1D                             ld R2, #RomBank2 /256
1535/     4E9 : 38 E1                             ld R3, R1
1536/     4EB : 92 32                             lde @RR2, R3            ; set EpromBank2, 3
1537/     4ED : AF                                ret
1538/     4EE :
1539/     4EE : 00 00               RomTable      DB 0, 0
1540/     4F0 : 00 00                             DB 0, 0
1541/     4F2 : 00 01                             DB 0, 1
1542/     4F4 : 01 00                             DB 1, 0
1543/     4F6 : 01 01                             DB 1, 1
1544/     4F8 :
1545/     4F8 :
1546/     4F8 :                     ;*****************************************************************
1547/     4F8 :                     ;
1548/     4F8 :                     ; Procedure: Bank_Ret
1549/     4F8 :                     ;
1550/     4F8 :                     ;  This procedure is used as the return linkage when another
1551/     4F8 :                     ;  procedure wishes to return to a location that is not in the same
1552/     4F8 :                     ;  external program bank.
1553/     4F8 :                     ;
1554/     4F8 :                     ; Inputs: none
1555/     4F8 :                     ;
1556/     4F8 :                     ; Outputs: none
1557/     4F8 :                     ;
1558/     4F8 :                     ; Algorithm:
1559/     4F8 :                     ;
1560/     4F8 :                     ;  Begin
1561/     4F8 :                     ;   Address.HiByte:=Stack[StackPtr]
1562/     4F8 :                     ;   Address.LoByte:=Stack[StackPtr-1]
1563/     4F8 :                     ;   LookUp_Rom(Address div 256)
1564/     4F8 :                     ;   Goto @Address
1565/     4F8 :                     ;  End
1566/     4F8 :                     ;
1567/     4F8 :                     ;*****************************************************************
1568/     4F8 :
1569/     4F8 : 70 FC               Bank_Ret      push FLAGS
1570/     4FA : 70 FD                             push RP
1571/     4FC : 31 30                             srp #Wrk_Sys2
1572/     4FE :                             assume RP:Wrk_Sys2
1573/     4FE : 18 FF                             ld R1, SPL              ; get location of return address
1574/     500 : 06 E1 02                          add R1, #2              ; account for the two pushes at
1575/     503 : E3 01                             ld R0, @R1              ;  entry of routine
1576/     505 : D6 04 BE                          call LookUp_Rom
1577/     508 : 50 FD                             pop RP
1578/     50A : 50 FC                             pop FLAGS
1579/     50C : AF                                ret
1580/     50D :
1581/     50D :
1582/     50D :
1583/     50D :                     ;*****************************************************************
1584/     50D :                     ; Module Int.Assem
1585/     50D :                     ;
1586/     50D :                     ; This module contains the routines associated with
1587/     50D :                     ; the various interrupts that the Widget controller deals with
1588/     50D :                     ;
1589/     50D :                     ;*****************************************************************
1590/     50D :
1591/     50D :                     ;*****************************************************************
1592/     50D :                     ;
1593/     50D :                     ; Procedure: Vector0:4 {illegal interrupts}
1594/     50D :                     ;
1595/     50D :                     ;*****************************************************************
1596/     50D :
1597/     50D :                     Vector0
1598/     50D :                     Vector1
1599/     50D :                     Vector2
1600/     50D :                     Vector3
1601/     50D : 50 E9               Vector4       pop R9                  ; pop FLAGS
1602/     50F : 50 EA                             pop R10                 ; pop return address
1603/     511 : 50 EB                             pop R11
1604/     513 : D6 05 1F                          call Abort
1605/     516 :
1606/     516 :
1607/     516 :                     ;*****************************************************************
1608/     516 :                     ;
1609/     516 :                     ; Procedure: Vector5 {IRQ5 interrupt}
1610/     516 :                     ;
1611/     516 :                     ;  This procedure decrements the DeadMan Timer Counter. If the
1612/     516 :                     ;  result is zero then Abort
1613/     516 :                     ;
1614/     516 :                     ;*****************************************************************
1615/     516 :
1616/     516 : 80 36               Vector5       decw Dmt_Counter
1617/     518 : 6B F3                             jr Z, Vector0           ; abort if zero
1618/     51A : 56 FA DF                          and IRQ, #0FFh-Timer1   ; clear old interrupt
1619/     51D : 9F                                ei
1620/     51E : BF                                iret
1621/     51F :
1622/     51F :
1623/     51F :
1624/     51F :                     ;*****************************************************************
1625/     51F :                     ; Module Except.Assem
1626/     51F :                     ;
1627/     51F :                     ;       PROCEDURE: Abort
1628/     51F :                     ;       PROCEDURE: Status_Call
1629/     51F :                     ;       PROCEDURE: Reset_StMach
1630/     51F :                     ;
1631/     51F :                     ;*****************************************************************
1632/     51F :
1633/     51F :                     ;*****************************************************************
1634/     51F :                     ;
1635/     51F :                     ; Procedure: Abort
```

```
1636/     51F :                     ;
1637/     51F :                     ;  This procedure is the 'garbage collector' for this set of
1638/     51F :                     ;  firmware. Abort is the place where all routines 'goto' if things
1639/     51F :                     ;  get so screwed up that they can no longer cope.
1640/     51F :                     ;
1641/     51F :                     ; Inputs: none, address of caller on stack
1642/     51F :                     ;
1643/     51F :                     ; Outputs: none
1644/     51F :                     ;
1645/     51F :                     ; Algorithm:
1646/     51F :                     ;
1647/     51F :                     ;  Begin
1648/     51F :                     ;   Except_Stat.Recovery:=false
1649/     51F :                     ;   DiskStat.Parked:=false
1650/     51F :                     ;   DiskStat.On_Track:=false
1651/     51F :                     ;   Clr_BankSwitch
1652/     51F :                     ;   Set_RamBank(Ram0)
1653/     51F :                     ;   Registers R14 and R15 get address of caller
1654/     51F :                     ;   For i:=0 To 15 Do
1655/     51F :                     ;    Abort_Stat[i]:=Register[Rp+i]
1656/     51F :                     ;   Initialize Write_BufferFence
1657/     51F :                     ;   SetStatus(Abort)
1658/     51F :                     ;   Goto RdLeave
1659/     51F :                     ;  End
1660/     51F :                     ;
1661/     51F :                     ;*********************************************************************
1662/     51F :
1663/     51F : 8F                  Abort           di
1664/     520 : 50 EE                               pop R14                 ; remember who called us
1665/     522 : 50 EF                               pop R15
1666/     524 : 70 FD                               push RP                 ; save context
1667/     526 : 31 40                               srp #Wrk_Scr
1668/     528 :                             assume RP:Wrk_Scr
1669/     528 : D6 02 00                            call Clr_BankSwitch
1670/     52B : 50 E0                               pop R0                  ; get caller's context from stack
1671/     52D : 1C 10                               ld R1, #16              ; load 16 locations
1672/     52F : EC 16                               ld R14, #Abort_Stat /256
1673/     531 : FC D8                               ld R15, #Abort_Stat #256
1674/     533 : 93 0E               Abort_Lp1       ldei @RR14, @R0
1675/     535 : 1A FC                               djnz R1, Abort_Lp1
1676/     537 : 31 10                               srp #Wrk_Sys
1677/     539 :                             assume RP:Wrk_Sys
1678/     539 : B0 FE                               clr SPH                 ; clean up stack
1679/     53B : E6 FF 80                            ld SPL, #Stack_Top
1680/     53E : D6 10 29                            call ExtStk_Vector
1681/     541 : D6 10 31                            call ZrRd_Vector
1682/     544 : EC 12                               ld R14, #WrBlkFence /256        ; re-write the write block fence
1683/     546 : FC 70                               ld R15, #WrBlkFence #256
1684/     548 : D6 10 19                            call LdPw_Vector
1685/     54B : 76 56 04                            tm DiskStat, #MultiBlk
1686/     54E : 6B 03                               jr Z, Abt_Stat_Ld
1687/     550 : D6 05 65                            call UpDate_Logical
1688/     553 : D6 05 80            Abt_Stat_Ld     call SS_Abort
1689/     556 : 76 02 08                            tm P2, #Bsy             ; check if BSY is set
1690/     559 : ED 10 13                            jp NZ, RdL_Vector
1691/     55C : 31 00                               srp #Wrk_Io
1692/     55E :                             assume RP:Wrk_Io
1693/     55E : AC 01                               ld R10, #Init_Response
1694/     560 : BC 00                               ld R11, #0
1695/     562 : 8D 02 78                            jp Clr_Bsy1
1696/     565 :                     ;
1697/     565 : D6 10 0D            UpDate_Logical  call LL_Vector
1698/     568 : 02 E5                               add R14, R5
1699/     56A : 16 ED 00                            adc R13, #0
1700/     56D : 16 EC 00                            adc R12, #0
1701/     570 : 0C 1C                               ld R0, #Wrk_Sys+12
1702/     572 : 1C 03                               ld R1, #3
1703/     574 : 2C 14                               ld R2, #LogicalBlock /256
1704/     576 : 3C A1                               ld R3, #LogicalBlock #256
1705/     578 : 93 02               UpD_Lgcl_Lp     ldei @RR2, @R0
1706/     57A : 1A FC                               djnz R1, UpD_Lgcl_Lp
1707/     57C : 56 56 FB                            and DiskStat, #0FFh-MultiBlk
1708/     57F : AF                                  ret
1709/     580 :
1710/     580 :
1711/     580 :                     ;*********************************************************************
1712/     580 :                     ;
1713/     580 :                     ; ProcedureSet: SetStatus_Calls
1714/     580 :                     ;
1715/     580 :                     ;  These are a few of the more commonly used SetStatus routines:
1716/     580 :                     ;
1717/     580 :                     ;   SS_Abort: SetStatus(Abort)
1718/     580 :                     ;              SetStatus(OperationFailed)
1719/     580 :                     ;
1720/     580 :                     ;   SS_OpFail: SetStatus(OperationFailed)
1721/     580 :                     ;
1722/     580 :                     ;   SS_ReadErr: SetStatus(ReadErrCount)
1723/     580 :                     ;
1724/     580 :                     ;   SS_NoHdr: SetStatus(NoHeaderFound)
1725/     580 :                     ;
1726/     580 :                     ;   SS_SprWarn: SetStatus(SprBlk_Warn)
1727/     580 :                     ;
1728/     580 :                     ;*********************************************************************
1729/     580 :
1730/     580 : 0C 01               SS_Abort        ld R0, #1               ; status byte 1
1731/     582 : 1C 01                               ld R1, #Stat_Abort
1732/     584 : D6 04 03                            call SetStatus
1733/     587 : 0C 00               SS_OpFail       ld R0, #0               ; status byte 0
1734/     589 : 1C 01                               ld R1, #Op_Failed
1735/     58B : D6 04 03            SS_Set          call SetStatus
1736/     58E : AF                                  ret
1737/     58F :
1738/     58F : 0C 03               SS_RdCnt        ld R0, #3               ; status byte 3
1739/     591 : 18 27                               ld R1, RdErrCnt
1740/     593 : 8B F6                               jr SS_Set
1741/     595 :
1742/     595 : 0C 00               SS_ReadErr      ld R0, #0               ; status byte 0
1743/     597 : 1C 08                               ld R1, #8
1744/     599 : D6 04 03                            call SetStatus
1745/     59C : 0C 03                               ld R0, #3               ; status byte 3
1746/     59E : 18 27                               ld R1, RdErrCnt
1747/     5A0 : 8B E9                               jr SS_Set
1748/     5A2 :
1749/     5A2 : 0C 00               SS_NoHdr        ld R0, #0               ; status byte 0
1750/     5A4 : 1C 04                               ld R1, #Stat_No_Hdr
1751/     5A6 : 8B E3                               jr SS_Set
1752/     5A8 :
1753/     5A8 : 0C 01               SS_SprWarn      ld R0, #1               ; status byte 1
```

```
1754/     5AA : 1C 20                              ld R1, #SprBlk_Warn
1755/     5AC : 8B DD                              jr SS_Set
1756/     5AE :
1757/     5AE :
1758/     5AE :                    ;*****************************************************************
1759/     5AE :                    ;
1760/     5AE :                    ; Procedure: Reset_StMach
1761/     5AE :                    ;
1762/     5AE :                    ;  This routine is used in the case where the state machine ended
1763/     5AE :                    ;  abnormally (i.e. it ended in a state other than its normal 'end-
1764/     5AE :                    ;  state'). The assumption is made that the drive clock is not
1765/     5AE :                    ;  available and that the controller must supply a signal in its
1766/     5AE :                    ;  place in order for the state machine to reset.
1767/     5AE :                    ;
1768/     5AE :                    ; Inputs: none
1769/     5AE :                    ;
1770/     5AE :                    ; Outputs: none
1771/     5AE :                    ;
1772/     5AE :                    ; Algorithm:
1773/     5AE :                    ;
1774/     5AE :                    ;  Begin
1775/     5AE :                    ;   Z8TestL:=true
1776/     5AE :                    ;   For i:=16 Downto 0 Do
1777/     5AE :                    ;    Zrwck:=true
1778/     5AE :                    ;    Zrwck:=false
1779/     5AE :                    ;   Z8TestL:=false
1780/     5AE :                    ;  End
1781/     5AE :                    ;
1782/     5AE :                    ;*****************************************************************
1783/     5AE :
1784/     5AE : 70 FD              Reset_StMach    push RP                 ; save context
1785/     5B0 : 31 00                              srp #Wrk_Io
1786/     5B2 :                           assume RP:Wrk_Io
1787/     5B2 : 56 E0 BF                           and P0, #0FFh-Not_Z8TestL       ; assert Z8TestL
1788/     5B5 : EC 10                              ld R14, #16             ; i:=16
1789/     5B7 : 46 E3 40           Res_StM_Lp      or P3, #Zrwck           ; set clock
1790/     5BA : 56 E3 BF                           and P3, #0FFh-Zrwck     ; clear clock
1791/     5BD : EA F8                              djnz R14, Res_StM_Lp
1792/     5BF : 46 E0 00                           or P0, #Z8TestL         ; deassert Z8TestL
1793/     5C2 : 46 E0 40                           or P0, #Zrwck           ; set clock
1794/     5C5 : 50 FD                              pop RP
1795/     5C7 : AF                                 ret
1796/     5C8 :
1797/     5C8 : 00 00              Int_Version     DB 00, 00
1798/     5CA :
1799/     5CA :
1800/     5CA :
1801/     5CA :                    ;*****************************************************************
1802/     5CA :                    ; Module Res.Misc.Assem
1803/     5CA :                    ;
1804/     5CA :                    ;       PROCEDURE ReSeek
1805/     5CA :                    ;       PROCEDURE New_Seek(Cylinder : WORD {RR12}
1806/     5CA :                    ;                          Head : BYTE {R13} Sector : BYTE {R14})
1807/     5CA :                    ;       PROCEDURE Load_Header(VAR BufferPtr : PTR {RR2})
1808/     5CA :                    ;       FUNCTION Load_Logical : LogicalBlock : 3 BYTES {R12:14}
1809/     5CA :                    ;       FUNCTION Chk_Chk_Byte(SourcePtr : PTR {RR14}
1810/     5CA :                    ;                          SourceLength : BYTE {RR8}) : BOOLEAN
1811/     5CA :                    ;       PROCEDURE Gen_Chk_Byte(SourcePtr : PTR {RR14}
1812/     5CA :                    ;                          SourceLength : BYTE {RR8})
1813/     5CA :                    ;       PROCEDURE ClearStatus
1814/     5CA :                    ;       PROCEDURE Ld_LgclBlk(Offset : BYTE {R0})
1815/     5CA :                    ;       PROCEDURE Set_SeekNeeded
1816/     5CA :                    ;
1817/     5CA :                    ;*****************************************************************
1818/     5CA :
1819/     5CA : E6 57 90           ReSeek          ld Seek_Type, #Access_Offset
1820/     5CD : 80 2A                              decw SeekCount          ; account for zero track seek
1821/     5CF : C8 52                              ld R12, Cylinder
1822/     5D1 : D8 53                              ld R13, Cylinder+1
1823/     5D3 : E8 54                              ld R14, Head
1824/     5D5 : F8 55                              ld R15, Sector
1825/     5D7 :
1826/     5D7 : 2C 10              New_Seek        ld R2, #Seek_Vector /256
1827/     5D9 : 3C 8D                              ld R3, #Seek_Vector #256
1828/     5DB : D6 04 AB                           call Bank_Call
1829/     5DE : 8D 04 F8                           jp Bank_Ret
1830/     5E1 :
1831/     5E1 :
1832/     5E1 :                    ;*****************************************************************
1833/     5E1 :                    ;
1834/     5E1 :                    ; Procedure: Load_Header
1835/     5E1 :                    ;
1836/     5E1 :                    ;  The procedure is responsible for generating (and loading
1837/     5E1 :                    ;  into the disk buffer) the header that will be used to identify
1838/     5E1 :                    ;  the correct block to operate on.
1839/     5E1 :                    ;
1840/     5E1 :                    ; Inputs: BufferPtr: PTR (RR2)
1841/     5E1 :                    ;
1842/     5E1 :                    ; Outputs: BufferPtr+5: PTR (RR2)
1843/     5E1 :                    ;
1844/     5E1 :                    ; Algorithm:
1845/     5E1 :                    ;
1846/     5E1 :                    ;  Begin
1847/     5E1 :                    ;   @BufferPtr(0):=HiCylinder
1848/     5E1 :                    ;   @BufferPtr(1):=LoCylinder
1849/     5E1 :                    ;   @BufferPtr(2)/bits 7:6:=Head
1850/     5E1 :                    ;   @BufferPtr(2)/bits 5:0:=Sector
1851/     5E1 :                    ;   @BufferPtr(3):=Invert(@BufferPtr(0))
1852/     5E1 :                    ;   @BufferPtr(4):=Invert(@BufferPtr(1))
1853/     5E1 :                    ;   @BufferPtr(4):=Invert(@BufferPtr(2))
1854/     5E1 :                    ;  End
1855/     5E1 :                    ;
1856/     5E1 :                    ;*****************************************************************
1857/     5E1 :
1858/     5E1 : 0C 52              Load_Header     ld R0, #Cylinder        ; load header: hi cylinder
1859/     5E3 : 93 02                              ldei @RR2, @R0
1860/     5E5 : 93 02                              ldei @RR2, @R0          ; load lo cylinder
1861/     5E7 : E3 10                              ld R1, @R0              ; head
1862/     5E9 : F0 E1                              swap R1
1863/     5EB : 90 E1                              rl R1
1864/     5ED : 90 E1                              rl R1
1865/     5EF : 0E                                 inc R0
1866/     5F0 : 43 10                              or R1, @R0              ; merge in sector
1867/     5F2 : 92 12                              lde @RR2, R1            ; load head/sector
1868/     5F4 : A0 E2                              incw RR2
1869/     5F6 : 08 52                              ld R0, Cylinder         ; Not(hi cylinder)
1870/     5F8 : 60 E0                              com R0
1871/     5FA : 92 02                              lde @RR2, R0
```

```
1872/     5FC : A0 E2                              incw RR2
1873/     5FE : 08 53                              ld R0, Cylinder+1       ; Not(lo cylinder)
1874/     600 : 60 E0                              com R0
1875/     602 : 92 02                              lde @RR2, R0
1876/     604 : A0 E2                              incw RR2
1877/     606 : 60 E1                              com R1                  ; Not(head/sector)
1878/     608 : 92 12                              lde @RR2, R1
1879/     60A : A0 E2                              incw RR2
1880/     60C : AF                                 ret
1881/     60D :
1882/     60D :
1883/     60D :                    ;*****************************************************************
1884/     60D :                    ;
1885/     60D :                    ; Function: LoadLogical
1886/     60D :                    ;
1887/     60D :                    ;  This function returns the last logical block number requested
1888/     60D :                    ;  by the host.
1889/     60D :                    ;
1890/     60D :                    ; Inputs: none
1891/     60D :                    ;
1892/     60D :                    ; Outputs: LogicalBlockNumber: 3 BYTES (R12:14)
1893/     60D :                    ;
1894/     60D :                    ;*****************************************************************
1895/     60D :
1896/     60D : 2C 14              Load_Logical    ld R2, #LogicalBlock /256
1897/     60F : 3C A1                              ld R3, #LogicalBlock #256
1898/     611 : 1C 03                              ld R1, #3
1899/     613 : 08 FD                              ld R0, RP
1900/     615 : 46 E0 0C                           or R0, #0Ch
1901/     618 : 83 02              Log_Log_Lp      ldei @R0, @RR2
1902/     61A : 1A FC                              djnz R1, Log_Log_Lp
1903/     61C : AF                                 ret
1904/     61D :
1905/     61D :
1906/     61D :                    ;*****************************************************************
1907/     61D :                    ;
1908/     61D :                    ; Function: Chk_Chk_Byte
1909/     61D :                    ;
1910/     61D :                    ;  This function tests the checkbyte of a servo command string.
1911/     61D :                    ;
1912/     61D :                    ; Inputs: SourcePtr: PTR (RR14)
1913/     61D :                    ;         SourceLength: BYTE (R8)
1914/     61D :                    ;
1915/     61D :                    ; Outputs: If CheckBytes match
1916/     61D :                    ;           Then Zero_Flag:=true
1917/     61D :                    ;           Else Zero_Flag:=false
1918/     61D :                    ;
1919/     61D :                    ; Algorithm:
1920/     61D :                    ;
1921/     61D :                    ;  Begin
1922/     61D :                    ;   Temp:=0
1923/     61D :                    ;   For i:=1 To SourceLength Do
1924/     61D :                    ;    Temp:=Temp+@SourcePtr
1925/     61D :                    ;    SourcePtr:=SourcePtr+1
1926/     61D :                    ;   Compare(Temp, @SourcePtr)
1927/     61D :                    ;  End
1928/     61D :                    ;
1929/     61D :                    ; Temp: R0, Scratch Register: R2
1930/     61D :                    ;
1931/     61D :                    ;*****************************************************************
1932/     61D :
1933/     61D : B0 E0              Chk_Chk_Byte    clr R0
1934/     61F : 82 2E              ChkB_Lp1        lde R2, @RR14
1935/     621 : 02 02                              add R0, R2
1936/     623 : A0 EE                              incw RR14
1937/     625 : 8A F8                              djnz R8, ChkB_Lp1
1938/     627 : 60 E0                              com R0
1939/     629 : 82 1E                              lde R1, @RR14
1940/     62B : A2 01                              cp R0, R1               ; set/clear zero flag
1941/     62D : AF                                 ret
1942/     62E :
1943/     62E :
1944/     62E :                    ;*****************************************************************
1945/     62E :                    ;
1946/     62E :                    ; Procedure: Gen_Chk_Byte
1947/     62E :                    ;
1948/     62E :                    ;  This function inserts the checkbyte into a servo command string
1949/     62E :                    ;
1950/     62E :                    ; Inputs: SourcePtr: PTR (RR14)
1951/     62E :                    ;         SourceLength: BYTE (R8)
1952/     62E :                    ;
1953/     62E :                    ; Outputs: none
1954/     62E :                    ;
1955/     62E :                    ; Algorithm:
1956/     62E :                    ;
1957/     62E :                    ;  Begin
1958/     62E :                    ;   Temp:=0
1959/     62E :                    ;   For i:=1 To SourceLength Do
1960/     62E :                    ;    Temp:=Temp+@SourcePtr
1961/     62E :                    ;    SourcePtr:=SourcePtr+1
1962/     62E :                    ;   Temp:=not(Temp)
1963/     62E :                    ;   @SourcePtr:=Temp
1964/     62E :                    ;  End
1965/     62E :                    ;
1966/     62E :                    ; Temp: R0, Scratch Register: R2
1967/     62E :                    ;
1968/     62E :                    ;*****************************************************************
1969/     62E :
1970/     62E : B0 E0              Gen_Chk_Byte    clr R0
1971/     630 : 82 2E              Gen_ChkB_Lp1    lde R2, @RR14
1972/     632 : 02 02                              add R0, R2
1973/     634 : A0 EE                              incw RR14
1974/     636 : 8A F8                              djnz R8, Gen_ChkB_Lp1
1975/     638 : 60 E0                              com R0
1976/     63A : 92 0E                              lde @RR14, R0
1977/     63C : AF                                 ret
1978/     63D :
1979/     63D :
1980/     63D :                    ;*****************************************************************
1981/     63D :                    ;
1982/     63D :                    ; Procedure: ClearStatus
1983/     63D :                    ;
1984/     63D :                    ;  This procedure clears all the bytes within the status array
1985/     63D :                    ;
1986/     63D :                    ; Inputs: none
1987/     63D :                    ;
1988/     63D :                    ; Outputs: none
1989/     63D :                    ;
```

```
1990/     63D :                     ; Algorithm:
1991/     63D :                     ;
1992/     63D :                     ;  Begin
1993/     63D :                     ;   For i:=1 To Length(StatusArray)
1994/     63D :                     ;     StatusArray[i]:=0
1995/     63D :                     ;  End
1996/     63D :                     ;
1997/     63D :                     ;*****************************************************************
1998/     63D :
1999/     63D : 2C 14               ClearStatus     ld R2, #CStatus0 /256
2000/     63F : 3C 95                               ld R3, #CStatus0 #256
2001/     641 : B0 E0                               clr R0                  ; clear all status bytes
2002/     643 : 1C 1C                               ld R1, #(End_CStatus-CStatus0)
2003/     645 : 92 02               Clr_Stat        lde     @RR2, R0
2004/     647 : A0 E2                               incw    RR2
2005/     649 : 1A FA                               djnz    R1, Clr_Stat
2006/     64B : 8D 04 F8                            jp      Bank_Ret
2007/     64E :
2008/     64E :
2009/     64E :                     ;*****************************************************************
2010/     64E :                     ;
2011/     64E :                     ; Procedure: Ld_LgclBlk {load logical block}
2012/     64E :                     ;
2013/     64E :                     ;  This procedure is used to extract the LogicalBlockNumber
2014/     64E :                     ;  information from a command string (sent by the host) and move it
2015/     64E :                     ;  into the global variable LogicalBlock.
2016/     64E :                     ;
2017/     64E :                     ; Inputs: Offset: BYTE (R0)
2018/     64E :                     ;
2019/     64E :                     ; Outputs: none
2020/     64E :                     ;
2021/     64E :                     ; Global Variables Changed: LogicalBlock
2022/     64E :                     ;
2023/     64E :                     ; Algorithm:
2024/     64E :                     ;
2025/     64E :                     ;  Begin
2026/     64E :                     ;   For i:=1 To Length (LogicalBlock) Do
2027/     64E :                     ;     LogicalBlock[i]:=CmdArray.KLogicalBlock[i]
2028/     64E :                     ;  End
2029/     64E :                     ;
2030/     64E :                     ;*****************************************************************
2031/     64E :
2032/     64E : 09 40               Ld_LgclBlk      ld ScrReg0, R0          ; pass parameter
2033/     650 : 70 FD                               push RP                 ; save context
2034/     652 : 31 40                               srp #Wrk_Scr
2035/     654 :                             assume RP:Wrk_Scr
2036/     654 : 2C 14                               ld R2, #CStatus4 /256
2037/     656 : 3C A5                               ld R3, #CStatus4 #256
2038/     658 : 02 30                               add R3, R0              ; add offset to BlockNumber
2039/     65A : 16 E2 00                            adc R2, #0
2040/     65D : EC 14                               ld R14, #LogicalBlock /256
2041/     65F : FC A1                               ld R15, #LogicalBlock #256
2042/     661 : 1C 04                               ld R1, #4               ; move 4 bytes
2043/     663 : 82 02               Ld_Lgcl_Lp      lde R0, @RR2
2044/     665 : 92 0E                               lde @RR14, R0
2045/     667 : A0 E2                               incw RR2
2046/     669 : A0 EE                               incw RR14
2047/     66B : 1A F6                               djnz R1, Ld_Lgcl_Lp
2048/     66D : 50 FD                               pop RP                  ; context switch
2049/     66F : 8D 04 F8                            jp Bank_Ret
2050/     672 :
2051/     672 :
2052/     672 :                     ;*****************************************************************
2053/     672 :                     ;
2054/     672 :                     ; Procedure: Set_SeekNeeded
2055/     672 :                     ;
2056/     672 :                     ;  This procedure acts in much the same fashion for the
2057/     672 :                     ;  LogicalBlockCache as DiskStat.On_Track does for simple seeks. It
2058/     672 :                     ;  keeps a legitimate seeks between two requests for the same block
2059/     672 :                     ;  number from returning a bogus result.
2060/     672 :                     ;
2061/     672 :                     ; Inputs: none
2062/     672 :                     ;
2063/     672 :                     ; Outputs: none
2064/     672 :                     ;
2065/     672 :                     ; Algorithm:
2066/     672 :                     ;
2067/     672 :                     ;  Begin
2068/     672 :                     ;   For i:=1 To CacheLength Do
2069/     672 :                     ;     CachStat[i].Seek.Needed:=true
2070/     672 :                     ;  End
2071/     672 :                     ;
2072/     672 :                     ;*****************************************************************
2073/     672 :
2074/     672 : 2C 16               Set_SeekNeeded  ld R2, #CacheStat /256
2075/     674 : 3C E8                               ld R3, #CacheStat #256
2076/     676 : 1C 13                               ld R1, #CacheLength-1
2077/     678 : 82 02               S_SeekN_Lp      lde R0, @RR2            ; get array value
2078/     67A : 46 E0 80                            or R0, #CachSeek
2079/     67D : 92 02                               lde @RR2, R0
2080/     67F : A0 E2                               incw RR2
2081/     681 : 1A F5                               djnz R1, S_SeekN_Lp
2082/     683 : 8D 04 F8                            jp Bank_Ret
2083/     686 :
2084/     686 :
2085/     686 : =>FALSE                     if $>ROMsize
2086/     686 :                                     error "\aROM size exceeded !!!"
2087/     686 : =>TRUE                      elseif
2088/     686 : FF FF FF FF FF FF                   DB ROMsize-$ 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 FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
               FF FF FF FF FF FF
```

```
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
                FF FF FF FF FF FF
2089/     800 : [2085]                      endif
2090/     800 :                             end
```

symbol table (* = unused):

```
 ABORT :                          51F C |  ABORT_LP1 :                      533 C |
 ABORT_STAT :                    16D8 C |  ABT_STAT_LD :                   553 C |
*ACCESS :                          80 - |  ACCESS_OFFSET :                 90 - |
*ACERASEL :                        20 - | *ACK_READ :                     302 C |
*ADD3 :                           3AC C |  APL_ACK :                       55 - |
*APL_EXCPT :                       0 - |  APPLE_MEM :                      0 - |
*ARCHITECTURE :  i386-unknown-win32 - | *B0_7_IO :                        0 - |
 B0_7_SER :                        40 - | *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_STATE :                       0 - |  BANKREG :                     1800 - |
 BANK_CALL :                      4AB C |  BANK_RET :                     4F8 C |
 BC_1 :                           4CE C | *BIGENDIAN :                      0 - |
*BLKLNDEX :                      1752 - |  BLKOFFSET :                   1751 - |
 BLKSTAT :                          5A - | *BLOCKID :                      200 - |
 BLOCKLENGTH :                    21D - | *BRANCHEXT :                      0 - |
 BSY :                              8 - |  BSY_LP2 :                      27E C |
*BUF2ARRAY :                     1274 C | *BUF2CRC :                     1489 C |
*BUF2ECC :                       148B C | *BUF2PW2 :                     1491 C |
*BUFDUMMY :                      1274 C | *BUFFER2 :                     1275 C |
*BUF_DAMAGE :                      20 - | *B_BLOCK :                        2 - |
*CACHEARRAY :                    16FC C |  CACHELENGTH :                   14 - |
 CACHESTAT :                     16E8 C |  CACHE_INDEX :                   5B - |
*CACHHDCHG :                       40 - |  CACHSEEK :                      80 - |
*CASESENSITIVE :                    0 - |  CHKB_LP1 :                     61F C |
*CHKB_MISMATCH :                    0 - | *CHK_CHK_BYTE :                 61D C |
*CHK_PARK1 :                      421 C |  CHK_PARK2 :                    427 C |
 CHK_PK3 :                        424 C | *CLEARBITMAP :                   20 - |
*CLEARSTATUS :                    63D C |  CLRNORMSTAT :                 1E5E - |
 CLRSTAT_VECTOR :                1039 - |  CLR_BANKSWITCH :               200 C |
*CLR_BSY :                        25F C |  CLR_BSY1 :                     278 C |
 CLR_BSY_RD :                     276 C |  CLR_B_LP :                     206 C |
*CLR_DMT :                        41F C |  CLR_STAT :                     645 C |
 CMD :                              4 - | *CMD_LEAVE :                    2AA C |
 CMD_LP2 :                        297 C |  CMD_LP3 :                      2BB C |
 CMD_NOTACK :                     2CC C | *CMD_TSTBSY :                   2A6 C |
*CMNDTYPE :                        F0 - | *CMND_EXCPT :                     1 - |
*CMND_LEN :                         7 - |  CMND_PENDING :                  80 - |
 CMND_PTR :                      101A - | *COMM_ERR :                       1 - |
*CONSTPI :        3.141592653589793 - | *CRCERRL :                       80 - |
*CRCSTAT :                         40 - |  CSTATUS0 :                    1495 C |
*CSTATUS1 :                      1499 C | *CSTATUS2 :                    149D C |
 CSTATUS3 :                      14A1 C |  CSTATUS4 :                    14A5 C |
*CSTATUS5 :                      14A9 C | *CSTATUS6 :                    14AD C |
*CUR_CYL :                         50 - |  CUR_THS :                     1754 - |
 CYLINDER :                        52 - | *DATARECAL :                      40 - |
 DATA_TYPE :                       58 - | *DATE :                    7/30/2013 - |
*DEC_BADCNT :                       2 - |  DEC_SCR_CNT :                 1EB C |
 DEC_SCR_END :                    1FF C | *DIAGNOSTIC :                     20 - |
*DIAG_CMNDS :                      13 - | *DIR_FRWD :                       20 - |
 DISKSTAT :                        56 - |  DISK_MEM :                       20 - |
*DISK_SPEED :                      20 - |  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 - |  DRWL_READ :                     80 - |
*DRWL_WRITE :                       0 - | *D_RDHDR_RESP :                   C - |
*D_READ_RESPONSE :                  B - | *D_R_ID_RESPONSE :                2 - |
*D_R_SPR_RESP :                     F - | *D_SCAN_RESPONSE :               15 - |
*D_WRITE_RESP :                     D - | *ECCERROR :                       0 - |
*ECCSTAT :                         80 - | *ECCTEST :                        2 - |
 END_CSTATUS :                   14B1 - |  END_WR_RESPONSE :                6 - |
*EPPOM0 :                           0 - | *EPROM1 :                         1 - |
 EPROM2 :                           2 - | *EPROM3 :                         3 - |
*EPROM4 :                           4 - |  EPROMOFFSET :                 1000 - |
 EPROMSIZE :                     1000 - | *EPROMSTARTADR :                  1 - |
 EPROMTEST :                      189 C |  EPROM_END :                    1C8 C |
 EPROM_FAIL :                      40 - |  EPROM_LP :                     1B4 C |
*ERROR :                           80 - |  EXCPT_STAT :                    24 - |
 EXTSTK_VECTOR :                 1029 - | *EXT_CLK :                        0 - |
*EX_BADBLOCK :                      4 - | *EX_CASE_MAX :                    A - |
*EX_HDRBAD :                        8 - | *EX_HDRSPR :                      A - |
*EX_READERR :                       6 - | *EX_SPRBLOCK :                    2 - |
*EX_UNDETERMINED :                  0 - |  E_LP :                         18B C |
*FALSE :                            0 - | *FBUF1CRC :                    1266 C |
*FBUF1ECC :                      1268 C | *FBUFFER1 :                    1052 C |
*FDATAGAP :                      1042 C |  FDATASYNC :                   1050 C |
*FENDGAP :                       126E C | *FHDRGAP :                     102A C |
 FHDRSYNC :                      103A C | *FHEADER :                     103C C |
 FLAGS :                           FC - | *FMENL :                          0 - |
 FMTBLK_1 :                       454 C | *FMTBLK_ABORT :                   7 - |
 FMTBLK_END :                     4A3 C |  FMTDELAY :                    1007 - |
 FMTERROR :                        80 - | *FMTLNTRL :                    14C4 C |
*FMTOFFSET :                     14C3 C |  FMTSRVOERR :                    40 - |
*FMTSUCCESS :                       20 - | *FMTTRK_ABORT :                   8 - |
*FMTTRK_POSERR :                    0 - |  FMT_NORM :                     48A C |
 FMT_RESIDENT :                   324 C | *FMT_RESPONSE :                  11 - |
 FMT_SERVOERR :                   496 C |  FMT_SRVOOK :                   49B C |
 FORMATARRAY :                   1020 C | *FORMATBLOCK :                  446 C |
*FOUND :                            1 - |  FPC_ELSE :                     338 C |
 FPC_END :                        33A C |  FREEP_LEAVE :                  436 C |
 FREE_PROC :                       69 - | *FREE_SLFTST :                   23 - |
 FREE_VECTOR :                   1021 - | *FRMTRECAL :                     70 - |
*FRST_SPRTBL :                      80 - | *FSCTRGAP :                    1020 C |
*FULLPMMU :                         1 - | *GATE_CLK :                      10 - |
 GEN_CHKB_LP1 :                   630 C | *GEN_CHK_BYTE :                 62E C |
*GET_WR_DATA :                    2ED C | *HAS64 :                          1 - |
*HASDSP :                           0 - | *HASFPU :                         0 - |
```

*HASPMMU :	0 -	*HDR_MISMATCH :	20 -
*HD_DIR_FRWD :	4 -	*HD_DIR_REV :	0 -
HEAD :	54 -	HEAP :	174E -
*HIMAXCYL :	2 -	*HIMAXLOGICAL :	0 -
HIRAMADR :	7FF -	HIREGADR :	7F -
*HIREVNUMBER :	1A -	*HISPR0 :	0 -
*HISPR1 :	0 -	HI_RWI_REG :	20 -
*HOME :	C0 -	*HOST_OVRFLOW :	12 -
*HS0 :	20 -	IBSY :	40 -
*ID_TYPE :	4 -	*ILLEGAL_BLOCK :	40 -
IMR :	FB -	*INC_BADCNT :	1 -
*INC_SPRCNT :	0 -	*INDEXMARK :	4 -
*INEXTMODE :	0 -	*INIT_HICYL :	1 -
*INIT_LOCYL :	F9 -	*INIT_PC :	C -
INIT_RESPONSE :	1 -	*INLWORDMODE :	0 -
*INMAXMODE :	0 -	*INSRCMODE :	0 -
*INSUPMODE :	0 -	*INTL_DFLT :	2 -
*INT_OUT :	C0 -	*INT_VEC0 :	0 C
*INT_VEC1 :	2 C	*INT_VEC2 :	4 C
*INT_VEC3 :	6 C	*INT_VEC4 :	8 C
*INT_VEC5 :	A C	*INT_VERSION :	5C8 C
INVERT_LED :	3ED C	IPR :	F9 -
IRQ :	FA -	*IRQ_INDEX :	1 -
*IRQ_SECDN :	2 -	*IRQ_SECTOR :	4 -
ISCAN_VECTOR :	1063 -	I_OREGUSED :	4 -
*I_SPR_RESPONSE :	12 -	*LBLK_BOUNDS :	2 -
LC_VECTOR :	1051 -	LDPW_VECTOR :	1019 -
*LD_LGCLBLK :	64E C	LD_LGCL_LP :	663 C
*LD_OFF_VAL :	20 -	LED :	1800 -
LEDSTAT :	1 -	LED_MASK :	FE -
*LED_WAIT :	3F4 C	LH_VECTOR :	1010 -
*LISTON :	1 -	LL_VECTOR :	100D -
*LOADSTATUS :	3FC C	*LOAD_HEADER :	5E1 C
*LOAD_LOGICAL :	60D C	LOGICALBLOCK :	14A1 -
LOG_LOG_LP :	618 C	*LOMAXCYL :	20 -
*LOMAXLOGICAL :	FF -	LOOKUP_ROM :	4BE C
*LOREVNUMBER :	45 -	*LOSPR0 :	55 -
*LOSPR1 :	AA -	LO_RWI_REG :	21 -
*LST_HEAD :	5E -	*LST_HICYL :	5C -
*LST_LOCYL :	5D -	*LST_SECTOR :	5F -
LU_ABORT :	4CB C	*MACEXP :	1 -
MAIN :	219 C	*MAIN_CACHE :	24E C
MAIN_EPROM :	224 C	*MAIN_LDSPR :	23E C
MAIN_LP1 :	254 C	MAIN_SELFTST :	22E C
MAIN_SET_R :	248 C	*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 -
*MSEL0 :	10 -	*MSEL1 :	20 -
MSWAIT :	1CB C	MSWAIT_1 :	1CD C
MSWAIT_LP :	1D0 C	MULTIBLK :	4 -
MULTIWR :	20 -	*NBRHDS :	2 -
NBRSCTRS :	13 -	*NBRTRACKS :	202 -
*NBRZONES :	10 -	*NESTMAX :	100 -
*NEW_SEEK :	5D7 C	*NIL :	80 -
*NOHDR_STATE :	0 -	*NON_RETRIG :	20 -
NORMFMT_STATE :	A -	*NORM_STATE :	2 -
*NOSPACE :	0 -	*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_IBSY :	2B3 C
NO_SPRTBL :	1 -	*NZERO_STAT :	8 -
*OFFSET :	10 -	*OFFSET_ON :	1 -
*OFF_AUTO :	40 -	*OFF_DIR_FRWD :	80 -
*OFF_DIR_REV :	0 -	*ON_TRACK :	80 -
*OPEN_DRAIN :	0 -	OP_FAILED :	1 -
*OVRLP_ABORT :	B -	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 -	PAR_OFF :	0 -
*PAR_ON :	80 -	PASSWORD :	1003 -
*PBLK :	1758 -	PBLOCK :	174E -
PC :	8 -	POWER_ON :	148 C
*POWER_RESET :	80 -	*PRE0 :	F5 -
PRE1 :	F3 -	*PROFILE :	0 -
*PRO_CMNDS :	2 -	*PRO_LOG_OFFSET :	1 -
*PSECTOR :	22 -	PWRFLG0 :	2C -
PWRFLG1 :	2D -	PWRFLG2 :	2E -
PWRFLG3 :	2F -	PWRRST :	10 -
PWRRST_LP :	137 C	*PWR_ON_RESET :	40 -
*RAM0 :	0 -	RAM1 :	1 -
*RAM2 :	2 -	*RAM3 :	3 -
RAMBANK0 :	1900 -	RAMLP1 :	161 C
RAMLP2 :	16E C	RAMLP3 :	178 C
RAMLPTIMES :	2 -	RAMOFFSET :	1000 -
RAMSIZE :	800 -	RAMTEST :	15D C
RAMTESTEXIT :	188 C	RAM_FAIL :	80 -
RAM_TABLE :	3D8 C	*RANDOM :	80 -
*RBUF1CRC :	122D C	*RBUF1ECC :	122F C
*RBUF1PW :	1236 C	RBUFFER1 :	1019 C
*RDATAGAP :	1011 C	*RDBLK_ABORT :	4 -
*RDCRCERR :	8 -	RDERRCNT :	27 -
*RDERROR :	80 -	*RDHDRH :	80 -
*RDHDRRECAL :	40 -	*RDHDR_ABORT :	5 -
*RDHDR_RESIDENT :	340 C	*RDHERROR :	80 -
*RDHSRVOERR :	40 -	*RDH_STAT_ARRAY :	1008 -
RDL_VECTOR :	1013 -	*RDNOHDRFND :	10 -
*RDSRVOERR :	40 -	RDSTAT :	26 -
*RDSUCCESS :	20 -	*RDUMMY :	1018 C
*RD_ABRT_RESP :	13 -	*RD_CMN_ABORT :	C -
RD_RES1 :	34A C	RD_RES2 :	34C C

*RD_RESIDENT :	346 C	*RD_SSTAT_RESP :	4 -
*RD_STAT_RESP :	3 -	READARRAY :	1000 C
*READHDRARRAY :	1000 C	*READSTATUS :	0 -
*READ_OP :	80 -	*READ_RESPONSE :	2 -
RECOVERY :	80 -	REGCOUNT :	80 -
REGLP1 :	70 C	REGLP2 :	76 C
REGLP3 :	7E C	REGLPTIMES :	2 -
REGUSED :	5 -	*RELAXED :	0 -
*RENDGAP :	1235 C	*RESEEK :	5CA C
RESET_STMACH :	5AE C	RES_ECC_ERR :	39B C
RES_STMACH :	393 C	RES_STM_LP :	5B7 C
*RETRIG :	30 -	*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 :	800 -
ROMTABLE :	4EE C	RP :	FD -
*RSCTRGAP :	1000 C	*RSTSRVO_RESP :	14 -
RWI :	4 -	*RWI_CYLINDER :	101 -
RWI_VALUE :	1008 -	*RW_FAIL :	2 -
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 -	*SD_S_C_RESP :	5 -
SECTDNL :	8 -	SECTOR :	55 -
SECTORMARK :	2 -	*SECTOR_CNT :	8 -
*SEEKCOMPLETE :	2 -	SEEKCOUNT :	2A -
*SEEK_ABORT :	D -	SEEK_TYPE :	57 -
SEEK_VECTOR :	108D -	*SEGPTRARRAY :	14C5 C
*SEQ_CACHSRCH :	8 -	*SERIAL_IN :	8 -
*SERIAL_OUT :	10 -	*SERR_NOTREADY :	0 -
SERVOERR :	10 -	SERVORDY :	20 -
*SERVORST :	0 -	*SERVO_DEAD :	0 -
*SERVO_FAIL :	10 -	*SERVO_LEN :	5 -
*SETBITMAP :	40 -	SETSTATUS :	403 C
SET_DMT :	413 C	SET_LED :	3E0 C
SET_RAMBANK :	3B3 C	SET_RB_START :	3BB C
*SET_RCVR_RESP :	8 -	SET_SEEKNEEDED :	672 C
*SINGLE_PASS :	0 -	*SIO :	F0 -
*SIORDY :	40 -	SLFTST_RESULT :	25 -
SLFTST_VECTOR :	1041 -	*SPARE :	10 -
*SPAREABORT :	9 -	SPAREARRAY :	14BB C
*SPAREBITMAP :	1547 C	*SPARECHECK :	1694 C
SPAREEND :	169A C	SPARELENGTH :	1DF -
*SPAREPW1 :	14BB C	*SPAREPW2 :	1696 C
*SPARETABLE :	1551 C	*SPARETMSTMP :	14BF C
SPH :	FE -	SPL :	FF -
*SPRBLK_HARD :	40 -	SPRBLK_WARN :	20 -
*SPRCNT_ABORT :	11 -	*SPRCOUNT :	1545 C
*SPRTBL_TYPE :	8 -	SPRTBL_VECTOR :	1049 -
*SPRTBL_WARN :	40 -	*SPRTHRESH :	3 -
*SRVOCMNDBUF :	14B1 C	*SRVO_R_ABORT :	A -
*SSTATUS0 :	14B6 C	SS_ABORT :	580 C
*SS_NOHDR :	5A2 C	*SS_OPFAIL :	587 C
*SS_RDCNT :	58F C	*SS_READERR :	595 C
SS_SET :	58B C	*SS_SPRWARN :	5A8 C
*STACKPTR :	174C C	*STACK_EXT :	0 -
STACK_IN :	4 -	STACK_TOP :	80 -
*START :	C C	STARTBLOCK :	174E -
*STARTL :	0 -	START_LOOP :	34 C
START_MASK :	9 -	*START_REGTEST :	38 C
*START_STATE :	0 -	START_STMACH :	350 C
START_VECTOR :	100A -	*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 -	*STDDEFZ8INC :	1 -
STK_HALT :	CC C	STK_RET :	CE C
STK_TEST :	B9 C	*STK_TEST1 :	BB C
*STRT_CMND :	0 -	*ST_AO_RESPONSE :	E -
*ST_MAP_RESPONSE :	E -	ST_RES_1 :	35A C
ST_RES_2 :	36F C	ST_RES_25 :	377 C
ST_RES_3 :	37B C	ST_RES_4 :	383 C
*SUB3 :	3A5 C	*SYS_CMNDS :	2 -
*SYS_LOG_OFFSET :	3 -	*SYS_RD_RESP :	22 -
*SYS_WREND_RESP :	27 -	*SYS_WREX_RESP :	A3 -
*SYS_WRVER_RESP :	24 -	*SYS_WR_RESP :	23 -
*S_BLOCK :	1 -	*S_CMND_BYTE :	0 -
*S_CMND_LEN :	5 -	*S_DIFF_BYTE :	1 -
*S_LOAD :	1 -	*S_NORM_STATUS :	1 -
*S_OFF_BYTE :	2 -	*S_PARK_RESP :	A -
*S_RATE_19_2 :	0 -	*S_RATE_57_6 :	80 -
*S_RSTR_RESPONSE :	7 -	*S_RST_ABORT :	3 -
S_SEEKN_LP :	678 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 -
*TESTBITMAP :	80 -	*TIME :	19:24:02 -
*TIMEOUT :	1 -	*TIMER0 :	10 -
TIMER1 :	20 -	TLBLOCK :	1751 -
TMR :	F1 -	*TOPOFSTACK :	17FF -
TOTEM_POL :	1 -	*TRUE :	1 -
*TST_HEAD :	0 -	*TST_HICYL :	2 -
*TST_LOCYL :	5 -	*TST_SCTR :	0 -
*UNUSED_REG :	59 -	UPDATE_LOGICAL :	565 C
UPD_LGCL_LP :	578 C	*USEABLE :	20 -
*USED :	40 -	USER_TYPE :	2 -
VECTOR0 :	50D C	VECTOR1 :	50D C
VECTOR2 :	50D C	VECTOR3 :	50D C
VECTOR4 :	50D C	VECTOR5 :	516 C
*VERSION :	142F -	*WAIT :	40 -

```
 WAIT_CMD :                    282 C |  WAIT_CMD1 :                   288 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 - |
 WPC_ELSE :                    320 C |  WPC_END :                    322 C |
 WRBLKFENCE :                 1270 C | *WRBLK_ABORT :                 6 - |
*WRBUF_OR :                     40 - |  WRERRCNT :                   29 - |
*WRERROR :                      80 - | *WRITEARRAY :                1000 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_RESPONSE :                4 - | *WR_CMN_ABORT :                 E - |
 WR_OP :                        20 - | *WR_RESIDENT :                30C C |
*WR_RESPONSE :                   3 - | *WR_SPR_RESP :                 10 - |
*WSCTRGAP :                   1000 C |  WSWAIT_DEC :                1E6 C |
*WT_CMD_NAK :                  2DC C |  W_10MB :                      1 - |
 W_20MB :                       0 - |  W_40MB :                      0 - |
*YMASK :                         F - |  Z8TESTL :                     0 - |
 Z8_APPLE :                     30 - |  Z8_MEM :                     10 - |
*ZONESHIFT :                     5 - | *ZONE_TABLE :                169A - |
 ZRRD_VECTOR :                1031 - |  ZRWCK :                       40 - |

    748 symbols
    453 unused symbols
```

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

STANDARD (0 changed characters)


0.11 seconds assembly time

   3096 lines source file
      2 passes
      0 errors
      0 warnings
```