```
SOURCE   FILE #01 =>PC
 INCLUDE FILE #02 =>PC.EQUATES
 INCLUDE FILE #03 =>PC.BOOTSPACE
 INCLUDE FILE #04 =>PC.BOOT
 INCLUDE FILE #05 =>PC.PACKET
 INCLUDE FILE #06 =>PC.CREAD
 INCLUDE FILE #07 =>PC.MAIN
0000:         0001     1 IIc        equ   1              ;Which machine?
0000:         0001     2 ROM        equ   1              ;RAM or ROM based
0000:         C000     3 TheOrg     equ   $C000
0000:         1000     4 version    equ   $1000
0000:                  5            lst   nou
0000:                  6 *
0000:         0001     7            ifeq  IIc
0000:                  9            else
```

```
0000:                11           fin
0000:                12 *
0000:          0001  13           X6502
0000:                14 *
0000:                15 *
0000:                16 *
0000:                17 *
0000:                18 *
0000:                19 ; PPPP  RRRR   000  TTTTT  000   CCC   000  L
0000:                20 ; P   P R   R 0   0   T   0   0 C   C 0   0 L
0000:                21 ; PPPP  RRRR  0   0   T   0   0 C     0   0 L
0000:                22 ; P     R R   0   0   T   0   0 C   C 0   0 L
0000:                23 ; P     R  R   000    T    000   CCC   000  LLLLL
0000:                24 ;
0000:                25 ;  CCC   000  N   N V   V EEEEE RRRR  TTTTT EEEEE RRRR
0000:                26 ; C   C 0   0 NN  N V   V E     R   R   T   E     R   R
0000:                27 ; C     0   0 N N N V   V EEEE  RRRR    T   EEEE  RRRR
0000:                28 ; C   C 0   0 N  NN  V V  E     R R     T   E     R R
0000:                29 ;  CCC   000  N   N   V   EEEEE R  R    T   EEEEE R  R
0000:                30 ;
```

```
0000:                32 *
0000:                33 *        UniDisk 3.5 Driver Firmware   Version 1.0
0000:                34 *
0000:                35 *    Written by Michael Askins   x6243   May 15, 1985
0000:                36 *
0000:                37 *           Copyright Apple Computer, Inc. 1985
0000:                38 *                   All Rights Reserved
0000:                39 *
0000:                40 *
0000:                41           MSB   ON
0000:                42 *
```

```
0000:  44 ************************************************************
0000:  45 *                                                          *
0000:  46 *  Modification History:                                   *
0000:  47 *                                                          *
0000:  48 * Rel  Date        Who  Action                              *
0000:  49 * ---------------------------------------- *
0000:  50 * ***  18 Dec 84   MSA  RELEASE VERSION 0.02 (Sony)         *
0000:  51 *      10 Jan 85   MSA  Added //c support:                  *
0000:  52 *                         General conditional assembly overhead *
0000:  53 *      16 Jan 85   MSA  Added retries and timeouts          *
0000:  54 *                       MSlot handled correctly            *
0000:  55 *                       Finished Boot code                 *
0000:  56 *                       Altered ProDOS errors - add $27 catchall *
0000:  57 *      18 Jan 85   MSA  Remove call to WAIT in monitor      *
0000:  58 *                       Add Boot failure messages          *
0000:  59 *      22 Jan 85   MSA  Add IWM reconfigure for //c version *
0000:  60 *      23 Jan 85   MSA  Move Comm routines to $C800 ($C900) *
0000:  61 *                       Fixed zero page preservation       *
0000:  62 * ***  23 Jan 85   MSA  RELEASE VERSION 0.03 (Apple)       *
0000:  63 *      25 Jan 85   MSA  Swap slot dep read and boot code (//c) *
0000:  64 *                       Add other //c differences...       *
0000:  65 *      30 Jan 85   MSA  Add auxtype byte                    *
0000:  66 *                       Fix comm error on receive packet   *
0000:  67 *                       Fix checksum to include MSBs of overhead *
0000:  68 *      07 Feb 85   MSA  Add COUT support on boot fail       *
0000:  69 * ***  08 Feb 85   MSA  RELEASE VERSION 1.00A (alpha)       *
0000:  70 *      22 Feb 85   MSA  Add bytecount in X,Y on PC calls    *
0000:  71 *                       Change hard reset time to 1 ms (was 83) *
0000:  72 *                       Crunched code by adding ClrPhases  *
0000:  73 *                       Add zeroing of third block byte (ProDOS) *
0000:  74 *      06 Mar 85   MSA  Fixed slot 7 goof (stack screw up)  *
0000:  75 *                       No clear phases on retries         *
0000:  76 *                       Hard reset time to 40 ms           *
0000:  77 *                       Pass #parms instead of unit# and no chk *
0000:  78 *                       Init code (all reset vs. comm reset) *
0000:  79 *                       Add 2 bytes to pass a full 9 byte cmd *
0000:  80 *      16 Mar 85   MSA  Fix bytecount on retries            *
0000:  81 *                       Boot block must be $800=$01, $801<>$00 *
0000:  82 *      17 Mar 85   MSA  Remove WRREQ while waiting for motor TO *
0000:  83 *                       Remove glitch on /ENBL2 in AssignID *
0000:  84 *      20 Mar 85   MSA  Add interrupt on/off/poll support   *
0000:  85 *                       Reset pulse to 80 ms               *
0000:  86 *                       //c delay of 100 ms on initial AssignID *
0000:  87 *                       ID bytes changed                   *
0000:  88 *                       Retransmit implemented (RecPack)   *
0000:  89 *                       Add send data packet retries (5)   *
0000:  90 *                       Rearrange PC stack adjust          *
0000:  91 *                       Add //c Appletalk vector           *
0000:  92 *      24 Mar 85   MSA  Add //c millisecond wait each call  *
0000:  93 * ***  25 Mar 85   MSA  RELEASE VERSION 1.00B (beta) (//e)  *
0000:  94 *      18 Apr 85   MSA  Clear decimal mode                  *
0000:  95 *                       Eight bytes are returned on stat unit#0 *
0000:  96 *                       Stat Unit#0 scode<>0 is rejected   *
0000:  97 *                       X and Y set to 0008 on status unit#0 *
0000:  98 *                       Enable interrupts done correctly   *
0000:  99 *                       Add unit#0 parameter count checking *
0000: 100 * ***  22 Apr 85   MSA  RELEASE VERSION 1.01B               *
0000: 101 * ***  15 May 85   MSA  RELEASE VERSION 1.0                 *
```

```
0000:              102 *                                                               *
0000:              103 ****************************************************************
0000:              104 *
0000:              105 *
0000:              106           include pc.equates
```

```
0000:                    2 *
0000:          00BF      3 PDIDByte  equ   $BF          ;ProDOS attributes byte
0000:          0000      4 PCID2     equ   $0          ;This means a Liron card
0000:                    5 *
0000:                    6 ************************
0000:                    7 *                          *
0000:                    8 *  Zero Page  (temps)      *
0000:                    9 *                          *
0000:                   10 ************************
0000:                   11 *
0000:                   12           dsect
0000:          0040     13 zeropage  equ   $0040
0040:          0040     14           org   zeropage
0040:                   15 *
0040:00                 16 checksum  dfb   0
0041:00                 17 topbits   dfb   0
0042:00                 18 CMDCode   dfb   0          ;ProDOS parameter passing area
0043:          0043     19 CMDPCount equ   *
0043:00                 20 CMDUnit   dfb   0
0044:          0044     21 CMDBuffer equ   *
0044:00                 22 CMDBufferl dfb  0
0045:00                 23 CMDBufferh dfb  0
0046:          0046     24 CMDSCode  equ   *
0046:          0046     25 CMDBlock  equ   *
0046:00                 26 CMDBlockl dfb   0
0047:00                 27 CMDBlockh dfb   0
0048:00                 28 CMDBlocks dfb   0
0049:00                 29 CMDSpare1 dfb   0
004A:00                 30 CMDSpare2 dfb   0
004B:          004B     31 rcvbuf    equ   *
004B:00                 32 grp7ctr   dfb   0
004C:00                 33 oddbytes  dfb   0
004D:          004D     34 statbyte  equ   *
004D:          004D     35 bytecount equ   *
004D:          004D     36 bytecountl equ  *
004D:          004D     37 next      equ   *
004D:00                 38 next1     dfb   0
004E:          004E     39 AuxType   equ   *
004E:          004E     40 bytecounth equ  *
004E:00                 41 next2     dfb   0
004F:          004F     42 RPacketType equ *
004F:00                 43 next3     dfb   0
0050:          0050     44 DeviceID  equ   *
0050:00                 45 next4     dfb   0
0051:          0051     46 HostID    equ   *
0051:00                 47 next5     dfb   0
0052:          0052     48 pointer   equ   *
0052:00                 49 next6     dfb   0
0053:00                 50 next7     dfb   0
0054:00 00              51 buffer    dw    0
0056:          0056     52 auxptr    equ   *
0056:00 00              53 buffer2   dw    0
0058:00                 54 slot      dfb   0
0059:          0059     55 temp      equ   *
0059:00                 56 tbodd     dfb   0
005A:00                 57 Unit      dfb   0          ;Current target unit
005B:00                 58 WPacketType dfb 0
005C:                   59 *
```

```
005C:                60 *
005C:          001C  61 ZPSize    equ   *-zeropage
005C:                62 *
005C:                63 *
0000:                64           dend
0000:                65 *
0000:          CFFF  66 ClearIOROMs equ $CFFF
0000:          0100  67 stack     equ   $100
0000:                68 *
0000:                69 *
0000:                70 ************************
0000:                71 *                      *
0000:                72 *   Screenhole Storage *
0000:                73 *                      *
0000:                74 ************************
0000:                75 *
0000:                76 * The screenhole layout is as follows:
0000:                77 *
0000:                78 *                //e          //c
0000:                79 *  ------------------
0000:                80 *  ProFlag     $478+n        $478
0000:                81 *  Retry       $4F8+n        $4F8
0000:                82 *  SHTemp1     $578+n        $578
0000:                83 *  SHTempX     $5F8+n        $5F8
0000:                84 *  SHTempY     $678+n        $678
0000:                85 *  Power1      $6F8+n        --
0000:                86 *  Power2      $778+n        --
0000:                87 *  NumDevices  $7F8+n        $6FE
0000:                88 *  SvBcL       $6F8          $6F8
0000:                89 *  SvBcH       $778          $778
0000:                90 *
0000:          0001  91           do    IIc
0000:          0473  92 scholes   equ   $473          ;Use the slot 0 sholes for temps
0000:                93           else
0000:                95           fin
0000:                96 *
0000:          0473  97 ProFlag   equ   scholes
0000:          04F3  98 Retry     equ   scholes+$80
0000:          0573  99 SHTemp1   equ   scholes+$100
0000:          0573 100 Retry2    equ   SHTemp1
0000:          05F3 101 SHTempX   equ   scholes+$180
0000:          0673 102 SHTempY   equ   scholes+$200
0000:          0001 103           ifeq  IIc
0000:               107           else
0000:          06F9 108 NumDevices equ  $6F9          ;Actually in slot 6
0000:               109           fin
0000:               110 *
0000:          06F8 111 SvBcL     equ   $6F8
0000:          0778 112 SvBcH     equ   $778
0000:               113 *
0000:          0025 114 cv        equ   $25
0000:          0024 115 ch        equ   $24
0000:          FC22 116 vtab      equ   $FC22
0000:          FDED 117 cout      equ   $FDED
0000:          07DB 118 bootscrn  equ   $7DB
0000:          07F8 119 MSlot     equ   $7F8
0000:          FE93 120 setvid    equ   $FE93
0000:          FE89 121 setkbd    equ   $FE89
```

```
0000:          FABA  122 AutoScan  equ   $FABA
0000:          E000  123 Basic     equ   $E000
0000:          0000  124 loc0      equ   $0          ;Boot parms
0000:          0001  125 loc1      equ   $1          ;
0000:                126 *
0000:          C797  127 SWPROTO   equ   $C797       ;//c bank switch to $C800
0000:          C784  128 SWRTS2    equ   $C784       ;RTS to bank 1
0000:                129 *
0000:                130 *
0000:                131 ************************
0000:                132 *                      *
0000:                133 *    General Equates   *
0000:                134 *                      *
0000:                135 ************************
0000:                136 *
0000:          00A5  137 PBBValue  equ   $A5         ;Powerup Byte Base Value
0000:          00FF  138 PBCValue  equ   $FF         ;Powerup Byte Complement Value
0000:                139 *
0000:          0000  140 PowerReset equ  $00
0000:          0080  141 CommReset equ   $80
0000:                142 *
0000:          0032  143 bsyto1    equ   50          ;(.55 ms) T/O on /BSY before send
0000:          000A  144 bsyto2    equ   10          ;(.12 ms) T/O on /BSY after send
0000:          001E  145 statmto   equ   30          ;30 bytes stat mark timeout
0000:          0009  146 cmdlength equ   9           ;Command packet length
0000:          00C3  147 packetbeg equ   $C3         ;Mark at beginning of packet
0000:          00C8  148 packetend equ   $C8         ;End of packet mark
0000:          0080  149 cmdmark   equ   $80         ;Command packet identifier
0000:          0081  150 statmark  equ   $81         ;Status Packet identifier
0000:          0082  151 datamark  equ   $82         ;Data Packet identifier
0000:                152 *
0000:          0007  153 iwmmode   equ   $07         ;No timer, asynch, latch
0000:                154 *
0000:          0000  155 SCDeviceStat equ 0          ;Get Device Specific Status
0000:          0001  156 SCGetDCB  equ   1           ;Get Dev Ctrl Block (modebits)
0000:          0002  157 SCRetNLStat equ 2           ;Return Newline Status
0000:          0003  158 SCGetDevInfo equ 3          ;Get Device Info Block
0000:                159 *
0000:          C080  160 iwm       equ   $C080
0000:                161 *
0000:          C080  162 reqclr    equ   iwm+0
0000:          C081  163 reqset    equ   iwm+1
0000:          C082  164 ca1clr    equ   iwm+2
0000:          C083  165 ca1set    equ   iwm+3
0000:          C084  166 ca2clr    equ   iwm+4
0000:          C085  167 ca2set    equ   iwm+5
0000:          C086  168 lstrbclr  equ   iwm+6
0000:          C087  169 lstrbset  equ   iwm+7
0000:          C088  170 monclr    equ   iwm+8
0000:          C089  171 monset    equ   iwm+9
0000:          C08A  172 enable1   equ   iwm+10
0000:          C08B  173 enable2   equ   iwm+11
0000:          C08C  174 l6clr     equ   iwm+12
0000:          C08D  175 l6set     equ   iwm+13
0000:          C08E  176 l7clr     equ   iwm+14
0000:          C08F  177 l7set     equ   iwm+15
0000:                178 *
0000:                179 *
```

```
0000:                 180 * Error0 codes
0000:                 181 *
0000:          0001   182 noanswer   equ   1
0000:          0002   183 nomark     equ   2
0000:          0004   184 wasreset   equ   4
0000:          0008   185 bytecmp    equ   8
0000:          0010   186 csumerr    equ   $10
0000:          0020   187 nopackend  equ   $20
0000:          0040   188 bushog     equ   $40
0000:                 189 *
0000:                 190 * Command Codes
0000:                 191 *
0000:          0000   192 StatusCmd  equ   $00
0000:          0001   193 ReadCmd    equ   $01
0000:          0002   194 WriteCmd   equ   $02
0000:          0003   195 FormatCmd  equ   $03
0000:          0004   196 ControlCmd equ   $04
0000:          0005   197 InitCmd    equ   $05
0000:                 198 *
0000:                 199 *
0000:          0040   200 Soft       equ   %01000000      ;The soft error bit in statbyte
0000:                 201 *
0000:          0001   202 BadCmd     equ   $01
0000:          0004   203 BadPCnt    equ   $04
0000:          0006   204 BusErr     equ   $06
0000:          0011   205 BadUnit    equ   $11
0000:          001F   206 NoInt      equ   $1F
0000:          0021   207 BadCtl     equ   $21
0000:          0022   208 BadCtlParm equ   $22
0000:          0027   209 IOError    equ   $27
0000:          0028   210 NoDrive    equ   $28
0000:          002B   211 WriteProt  equ   $2B
0000:          002D   212 BadBlock   equ   $2D
0000:          002F   213 OffLine    equ   $2F
0000:          0068   214 LastOne    equ   Soft+NoDrive
0000:          0067   215 SoftError  equ   Soft+IOError
0000:                 216 *
0000:          0010   217 SVMask1    equ   $10
0000:                 218 *
0000:          0BB8   219 RC1        equ   3000           ;Send a command pack 3000 times (3 sec)
0000:          0005   220 RC2        equ   5              ;Data Packs (sent/rcd) get tried only 5
                                                           times
0000:                 221 *
0000:                 222 *
0000:                 107 *
0000:          0001   108            do    IIc^ROM        ;If //c ROM start is $C500
--- NEXT OBJECT FILE NAME IS CPC.0
C500:          C500   109            org   $C500
C500:                 110            else
C500:                 116            fin
C500:                 117 *
C500:                 118            include pc.bootspace
```

```
C500:                2 *lst off
C500:                3 *
C500:        0001    4          ifeq  IIc^ROM       ;If NOT the //c ROM version, do this
C500:              937          fin
C500:              938 *
```

```
C500:                 940 *
C500:          0001   941           do    IIc
C500:          0060   942 TheOff     equ   $60            ;On //c IWM in slot 6
C500:                 943           else
C500:                 945           fin
C500:                 946 *
C500:                 947 *lst on
C500:                 948 *
C500:                 949 * Here beginneth that code which resideth in the boot space
C500:                 950 *  at the time the card resteth in slot the fifth.
C500:                 951 *
C500:          C500   952 C500org    equ   *
C500:                 953 *
C500:                 954 * Auto Boot signature bytes
C500:                 955 *  This is also the boot (auto & PR#5) entry point.
C500:                 956 *
C500:A2 20            957           ldx   #$20
C502:A2 00            958           ldx   #$00
C504:A2 03            959           ldx   #$03
C506:                 960 *
C506:C9 00            961           cmp   #0             ;Flag that this is a boot
C508:          0001   962           do    IIc^ROM
C508:B0 17   C521     963           bcs   BootC
C50A:                 964           else
C50A:                 966           fin
C50A:                 967 *
C50A:                 968 * Here is the ProDOS normal entry point
C50A:                 969 *
C50A:          C50A   970 ProDOSEntry equ *
C50A:                 971 *
C50A:                 972 * Set up so that ProFLAG will have the top bit set
C50A:                 973 *
C50A:38               974           sec
C50B:B0 01   C50E     975           bcs   *+3              ;Skip the clear
C50D:                 976 *
C50D:                 977 * This is the MLIxface entry point
C50D:                 978 *
C50D:          C50D   979 MLIEntry   equ   *              ;Only use this label in //c version
C50D:18               980           clc
C50E:A2 05            981           ldx   #$05
C510:7E 73 04         982           ror   ProFLAG,x      ;ProFLAG[7]=1 if ProDOS, =0 if MLI
C513:18               983           clc                  ;This is not a boot entry
C514:                 984 *
C514:                 985 * Now save mslot and clear all $C800 ROMs
C514:                 986 *
C514:          C514   987 bootcase5 equ   *
C514:A2 C5            988           ldx   #$C5           ;Load value for MSLOT
C516:8E F8 07         989           stx   MSlot
C519:A2 05            990           ldx   #$05
C51B:AD FF CF         991           lda   ClearIOROMs    ;Clear all $C800 latches but ours
C51E:                 992 *
C51E:          0001   993           do    IIc^ROM
C51E:4C 97 C7         994           jmp   SWPROTO
C521:          C521   995 BootC      equ   *
C521:A2 05            996           ldx   #$05           ;Need slot number
C523:                 997           else
C523:                1189           fin
C523:                1190 *
```

```
C523:              1191 * lst off
C523:              1192 *
C523:        0001  1193           ifeq  IIc^ROM       ;If not the //c ROM, more boot spaces
C523:              1658           fin
C523:              1659 *lst on
C523:               119 *
C523:        0001   120           do    IIc^ROM
C523:               121           include pc.boot
```

```
C523:                  2 *
C523:          C523    3 Bootcode  equ   *
C523:86 58             4           stx   slot
C525:                  5 *
C525:          0001    6           do    IIc^ROM
C525:A9 C5             7           lda   #$C5
C527:8D F8 07          8           sta   MSlot
C52A:20 76 C5          9           jsr   reset
C52D:                 10           else
C52D:                 14           fin
C52D:                 15 *
C52D:A0 05            16           ldy   #5            ;Copy a command table
C52F:B9 70 C5         17 bc1       lda   boottab,y
C532:99 42 00         18           sta   cmdcode,y
C535:88               19           dey
C536:10 F7    C52F    20           bpl   bc1
C538:                 21 *
C538:                 22 * Now on //e, patch the Unit number (slot*16)
C538:                 23 *
C538:          0001   24           ifeq  IIc^ROM
C538:                 31           fin
C538:                 32 *
C538:                 33 * Now do the read from block zero
C538:                 34 *
C538:          0001   35           do    IIc^ROM
C538:20 0A C5         36           jsr   ProDOSEntry
C53B:                 37           else
C53B:                 39           fin
C53B:B0 15    C552    40           bcs   bootfail      ;If fail, check loc
C53D:                 41 *
C53D:AE 00 08         42           ldx   $800          ;If ($800)<>1 this is no A// boot disk
C540:CA               43           dex
C541:D0 0F    C552    44           bne   bootfail
C543:                 45 *
C543:AE 01 08         46           ldx   $801          ;If $801 is zero, no boot
C546:F0 0A    C552    47           beq   bootfail
C548:                 48 *
C548:                 49 * It all looks okay.  Jump to the code with N0 in X.
C548:                 50 *
C548:A5 58            51           lda   Slot
C54A:0A               52           asl   a
C54B:0A               53           asl   a
C54C:0A               54           asl   a
C54D:0A               55           asl   a
C54E:AA               56           tax
C54F:4C 01 08         57           jmp   $801          ;Jump to it
C552:                 58 *
C552:                 59 * Do this code if the boot can't be done.
C552:                 60 *  If this was an autoboot (loc=$CN00), continue the slot scan.
C552:                 61 *  If not, drop into basic after issuing appropriate message
C552:                 62 *
C552:                 63 *
C552:          C552   64 bootfail  equ   *
C552:                 65 *
C552:          0001   66           do    IIc
C552:A2 10            67           ldx   #>bmsglen-1
C554:          C554   68 morchrs   equ   *
C554:BD 5F C5         69           lda   bootmsg,x
```

```
C557:9D DB 07       70            sta   bootscrn,x
C55A:CA             71            dex
C55B:10 F7   C554   72            bpl   morchrs
C55D:80 FE   C55D   73 coma       bra   coma          ;He's dead Jim.
C55F:               74 *
C55F:C3 E8 E5 E3    75 bootmsg    asc   'Check        Disk Drive.'
C570:       0011    76 bmsglen    equ   *-bootmsg
C570:               77            else
C570:              131            fin
C570:              132 *
C570:01 50 00 08   133 boottab    dfb   ReadCMD,$50,0,8,0,0 ;Read from 1st; blk0->$801
C576:              134 *
C576:              135 *
C576:              136 * This routine is called from the //c reset code.  It forces a
C576:              137 *  reset of the PC Bus.
C576:              138 *
C576:       0001   139            do    IIc^ROM
C576:       C576   140 Reset      equ   *
C576:A2 08         141            ldx   #8
C578:       C578   142 rst1       equ   *
C578:BD 83 C5      143            lda   rcode,x
C57B:95 00         144            sta   loc0,x
C57D:CA            145            dex
C57E:10 F8   C578  146            bpl   rst1
C580:4C 00 00      147            jmp   loc0
C583:              148 *
C583:       C583   149 rcode      equ   *
C583:20 0D C5      150            jsr   MLIEntry
C586:05            151            dfb   InitCMD
C587:07 00         152            dw    $0007
C589:60            153            rts
C58A:              154 *
C58A:01 00         155 cmdlist   dfb   1,0           ;One parm - the unit $00
C58C:              156            fin
C58C:              157 *
C58C:              158 *
--- NEXT OBJECT FILE NAME IS CPC.1
C5F5:       C5F5   122            org   $C5F5
C5F5:4C 52 C5      123            jmp   bootfail      ;Jump to the boot failure message
C5F8:4C 76 C5      124            jmp   reset         ;Reset vector
C5FB:00            125            dfb   PCID2
C5FC:00 00         126            dw    0
C5FE:BF            127            dfb   PDIDByte
C5FF:0A            128            dfb   >ProDOSEntry
C600:              129 *
--- NEXT OBJECT FILE NAME IS CPC.2
C880:       C880   130            org   $C880
C880:4C 4B CD      131            jmp   Entry         ;The //c bank switch jumps here
C883:4C E8 CF      132            jmp   AppleTalkEntry
C886:              133            fin
C886:              134 *
C886:              135            include pc.packet
C886:                   1         lst   cyc
C886:                   2 *
```

```
C886:                    4 ************************************************************
C886:                    5 *                                                          *
C886:                    6 *  SendOnePack                    Send a CBus Packet      *
C886:                    7 *                                                          *
C886:                    8 *    This routine sends a packet of data across the      *
C886:                    9 *  bus.  The protocol is as follows:                       *
C886:                   10 *                  ----------------------------          *
C886:                   11 *  REQ _______|2                          5|_________  *
C886:                   12 *               ---------------------------------        *
C886:                   13 * /BSY ___|1              3              4|__________  *
C886:                   14 *                                                         *
C886:                   15 *    1)  Device signals ready for data                    *
C886:                   16 *    2)  Host signals data iminent                        *
C886:                   17 *    3)  Packet is transmitted (sync, command mark,       *
C886:                   18 *         ids, contents, checksum [msb=1])                *
C886:                   19 *    4)  Device signals packet recieved                   *
C886:                   20 *    5)  Host finishes send data cycle                    *
C886:                   21 *                                                         *
C886:                   22 *    The bytes are sent in slow mode (32 cycles/byte)     *
C886:                   23 *  and the timing is critical.  Branches which should     *
C886:                   24 *  not cross page boundaries are marked.                  *
C886:                   25 *                                                         *
C886:                   26 *   Input:  buffer (2 bytes) <- ptr to data to send      *
C886:                   27 *           bytecount (2)    <- length (bytes) of data   *
C886:                   28 *           packettype (1)   <- command or data packet   *
C886:                   29 *           CMDUnit (1)      <- # of device to receive   *
C886:                   30 *                                                         *
C886:                   31 *   Output: carry set- handshake error                   *
C886:                   32 *                   clr- bytes sent                      *
C886:                   33 *                                                         *
C886:                   34 ************************************************************
C886:                   35 *
C886:         C886      36 SendOnePack equ *
C886:                   37 *
C886:                   38 * Prep for the transmission
C886:                   39 *
C886:20 64 CB    (6)    40         jsr   WritePrep      ;Does a bunch of stuff
C889:                   41 *
C889:                   42 * Enable PC chain.
C889:                   43 *
C889:20 80 CA    (6)    44         jsr   enablechain    ;This sets X reg
C88C:A0 07       (2)    45         ldy   #iwmmode       ;This is the mode value
C88E:20 1F CC    (6)    46         jsr   SetIWMode      ;Don't mess unless we gotta
C891:                   47 *
C891:                   48 * Turn on the IWM
C891:                   49 *
C891:BD 8B C0    (4)    50         lda   enable2,x      ;Don't disturb //c internal drive
C894:BD 89 C0    (4)    51         lda   monset,x
C897:                   52 *
C897:                   53 * Loop until the chain becomes unbusy
C897:                   54 *
C897:A0 32       (2)    55         ldy   #bsyto1        ;Each loop is 11 microseconds
C899:BD 8E C0    (4)    56 ubsy1   lda   17clr,x        ;Test if /BSY is hi or lo
C89C:30 07    C8A5(3)   57         bmi   chainunbsy     ;If hi, bus is not busy
C89E:88          (2)    58         dey
C89F:D0 F8    C899(3)   59         bne   ubsy1          ;Keep trying
C8A1:                   60 *
C8A1:38          (2)    61         sec
```

```
C8A2:4C CF C9    (3)   62          jmp   sd10
C8A5:                  63 *
C8A5:                  64 * Tell the bus that data is coming and send the sync bytes
C8A5:                  65 *  Sync is groups of eight 2's separated by a 6 (micS cell)
C8A5:                  66 *  (11111111001111111100111111100 ...)
C8A5:                  67 *
C8A5:        C8A5      68 chainunbsy equ *
C8A5:BD 81 C0    (4)   69          lda   reqset,x      ;Raise REQ
C8A8:                  70 *
C8A8:A0 05       (2)   71          ldy   #5            ;Sync plus packet begin
C8AA:                  72 *
C8AA:A9 FF       (2)   73          lda   #$FF          ;Send out the 1st byte sync
C8AC:9D 8F C0    (5)   74          sta   17set,x
C8AF:                  75 *
C8AF:B9 D6 C9    (4)   76 ssb      lda   preamble,y
C8B2:                  77 *
C8B2:                  78 *
C8B2:1E 8C C0    (7)   79 ssd      asl   16clr,x       ;Wait 'til buffer empty
C8B5:90 FB   C8B2(3)   80          bcc   ssd
C8B7:                  81 *
C8B7:9D 8D C0    (5)   82          sta   16set,x
C8BA:88          (2)   83          dey
C8BB:10 F2   C8AF(3)   84          bpl   ssb           ;Back for more bytes
C8BD:                  85 *
C8BD:                  86 * Send over the desination ID
C8BD:                  87 *
C8BD:A5 5A       (3)   88          lda   Unit
C8BF:09 80       (2)   89          ora   #$80          ;Make the device ID
C8C1:20 53 CA    (6)   90          jsr   sendbyte
C8C4:                  91 *
C8C4:                  92 * Send the source ID (that's us... we're an $80)
C8C4:                  93 *
C8C4:20 51 CA    (6)   94          jsr   send80
C8C7:                  95 *
C8C7:                  96 * Send over the packet type (command or data)
C8C7:                  97 *
C8C7:A5 5B       (3)   98          lda   Wpackettype
C8C9:20 53 CA    (6)   99          jsr   sendbyte
C8CC:                 100 *
C8CC:                 101 * Send the Auxilliary Type byte (an $80 from this rev PC)
C8CC:                 102 *
C8CC:20 51 CA    (6)  103          jsr   send80
C8CF:                 104 *
C8CF:                 105 * Send the status byte (null for us), and length bytes
C8CF:                 106 *
C8CF:20 51 CA    (6)  107          jsr   send80
C8D2:A5 4C       (3)  108          lda   oddbytes
C8D4:09 80       (2)  109          ora   #$80
C8D6:20 53 CA    (6)  110          jsr   sendbyte
C8D9:A5 4B       (3)  111          lda   grp7ctr
C8DB:09 80       (2)  112          ora   #$80
C8DD:20 53 CA    (6)  113          jsr   sendbyte
C8E0:                 114 *
C8E0:                 115 * Now send the "oddbytes" part of the packet contents
C8E0:                 116 *
C8E0:A5 4C       (3)  117          lda   oddbytes      ;Get # of "odd" bytes
C8E2:F0 15   C8F9(3)  118          beq   sob2          ;Skip if no odd bytes
C8E4:                 119 *
```

```
C8E4:A0 FF      (2)  120         ldy   #$FF
C8E6:A5 59      (3)  121         lda   tbodd          ;Get the odd bytes msb's (A[7]=1)
C8E8:                122 *
C8E8:1E 8C C0   (7)  123 sob1    asl   l6clr,x       ;Do a write handshake
C8EB:90 FB  C8E8(3)  124         bcc   sob1
C8ED:9D 8D C0   (5)  125         sta   l6set,x
C8F0:C8         (2)  126         iny
C8F1:B1 54      (5)  127         lda   (buffer),y    ;Get the data byte
C8F3:09 80      (2)  128         ora   #$80          ;Flip on the hi bit
C8F5:C4 4C      (3)  129         cpy   oddbytes      ;Are we done?
C8F7:90 EF  C8E8(3)  130         blt   sob1
C8F9:                131 *
C8F9:                132 * Now send over the groups of seven contents
C8F9:                133 * Currently assume there must be at least one group of 'em
C8F9:                134 *
C8F9:       C8F9     135 sob2    equ   *
C8F9:A5 4B      (3)  136         lda   grp7ctr       ;Check if there are groups to send
C8FB:D0 03  C900(3)  137         bne   sob3          ;=> At least one group
C8FD:4C 99 C9   (3)  138         jmp   datdone       ;Skip to send checksum
C900:                139 *
C900:       C900     140 sob3    equ   *
C900:EA         (2)  141         nop                 ;Waste 2 cycles
C901:A0 00      (2)  142         ldy   #0
C903:A5 41      (3)  143 start   lda   topbits
C905:9D 8D C0   (5)  144         sta   l6set,x
C908:                145 *
C908:                146 * Send first byte
C908:                147 *
C908:A5 4D      (3)  148         lda   next1
C90A:09 80      (2)  149         ora   #$80
C90C:84 59      (3)  150         sty   temp          ;Swap Y for short handshake
C90E:BC 8C C0   (4)  151 ache1   ldy   l6clr,x       ;Wait 'til buffer ready
C911:10 FB  C90E(3)  152         bpl   ache1
C913:9D 8D C0   (5)  153         sta   l6set,x       ;Send the byte
C916:A4 59      (3)  154         ldy   temp          ;Get back Y
C918:                155 *
C918:                156 * Prep the next "1st" byte for next time
C918:                157 *
C918:B1 56      (5)  158         lda   (buffer2),y
C91A:85 4D      (3)  159         sta   next1
C91C:0A         (2)  160         asl   a
C91D:26 41      (5)  161         rol   topbits       ;Store the top bit
C91F:C8         (2)  162         iny                 ;Next byte
C920:                163 *
C920:                164 * It's possible that we're at a page boundary now. If so, bump the
C920:                165 *  hi order part of the pointer.
C920:                166 *
C920:D0 05  C927(3)  167         bne   skip1
C922:E6 57      (5)  168         inc   buffer2+1
C924:4C 29 C9   (3)  169         jmp   skip2
C927:48         (3)  170 skip1   pha                 ;Equalize the cases
C928:68         (4)  171         pla
C929:                172 *
C929:                173 * Push us ahead by an additional 8 cycles for margin reasons
C929:                174 *  Plus I gotta get the topbits MSB set somehow...
C929:                175 *
C929:       C929     176 skip2   equ   *
C929:A9 02      (2)  177         lda   #%00000010    ;Flip what will be MSB
```

```
C92B:05 41      (3)  178         ora   topbits
C92D:85 41      (3)  179         sta   topbits
C92F:                180 *
C92F:                181 * Send the second byte
C92F:                182 *
C92F:A5 4E      (3)  183         lda   next2
C931:09 80      (2)  184         ora   #$80
C933:9D 8D C0   (5)  185         sta   16set,x       ;Send the byte
C936:B1 56      (5)  186         lda   (buffer2),y
C938:85 4E      (3)  187         sta   next2
C93A:0A         (2)  188         asl   a
C93B:26 41      (5)  189         rol   topbits       ;Store the top bit
C93D:C8         (2)  190         iny                 ;Next byte
C93E:                191 *
C93E:                192 * Send the third byte
C93E:                193 *
C93E:A5 4F      (3)  194         lda   next3
C940:09 80      (2)  195         ora   #$80
C942:9D 8D C0   (5)  196         sta   16set,x       ;Send the byte
C945:B1 56      (5)  197         lda   (buffer2),y
C947:85 4F      (3)  198         sta   next3
C949:0A         (2)  199         asl   a
C94A:26 41      (5)  200         rol   topbits       ;Store the top bit
C94C:C8         (2)  201         iny                 ;Next byte
C94D:                202 *
C94D:                203 * Send the fourth byte
C94D:                204 *
C94D:A5 50      (3)  205         lda   next4
C94F:09 80      (2)  206         ora   #$80
C951:9D 8D C0   (5)  207         sta   16set,x       ;Send the byte
C954:B1 56      (5)  208         lda   (buffer2),y
C956:85 50      (3)  209         sta   next4
C958:0A         (2)  210         asl   a
C959:26 41      (5)  211         rol   topbits       ;Store the top bit
C95B:C8         (2)  212         iny                 ;Next byte
C95C:                213 *
C95C:                214 * After the first 256 bytes, we will cross pages here.  If we did
C95C:                215 *  cross, bump the buffer pointer. If not, equalize the cases with
C95C:                216 *  seven cycles of time wasting.
C95C:                217 *
C95C:D0 05   C963(3) 218         bne   skip3
C95E:E6 57      (5)  219         inc   buffer2+1
C960:4C 65 C9   (3)  220         jmp   skip4
C963:48         (3)  221 skip3   pha
C964:68         (4)  222         pla
C965:        C965    223 skip4   equ   *
C965:                224 *
C965:                225 * Send the fifth byte
C965:                226 *
C965:A5 51      (3)  227         lda   next5
C967:09 80      (2)  228         ora   #$80
C969:9D 8D C0   (5)  229         sta   16set,x       ;Send the byte
C96C:B1 56      (5)  230         lda   (buffer2),y
C96E:85 51      (3)  231         sta   next5
C970:0A         (2)  232         asl   a
C971:26 41      (5)  233         rol   topbits       ;Store the top bit
C973:C8         (2)  234         iny                 ;Next byte
C974:                235 *
```

```
C974:                   236 * Send the sixth byte
C974:                   237 *
C974:A5 52        (3)   238         lda   next6
C976:09 80        (2)   239         ora   #$80
C978:9D 8D C0     (5)   240         sta   l6set,x       ;Send the byte
C97B:B1 56        (5)   241         lda   (buffer2),y
C97D:85 52        (3)   242         sta   next6
C97F:0A           (2)   243         asl   a
C980:26 41        (5)   244         rol   topbits       ;Store the top bit
C982:C8           (2)   245         iny                 ;Next byte
C983:                   246 *
C983:                   247 * Send the last byte of the group
C983:                   248 *
C983:A5 53        (3)   249         lda   next7
C985:09 80        (2)   250         ora   #$80
C987:9D 8D C0     (5)   251         sta   l6set,x       ;Send the byte
C98A:B1 56        (5)   252         lda   (buffer2),y
C98C:85 53        (3)   253         sta   next7
C98E:0A           (2)   254         asl   a
C98F:26 41        (5)   255         rol   topbits       ;Store the top bit
C991:C8           (2)   256         iny                 ;Next byte
C992:                   257 *
C992:                   258 * Now see if we have sent enough groups of seven
C992:                   259 *
C992:C6 4B        (5)   260         dec   grp7ctr
C994:F0 03   C999(3)    261         beq   datdone
C996:                   262 *
C996:                   263 * Otherwise, back to do more.  Note it's too far for a branch.
C996:                   264 *
C996:4C 03 C9     (3)   265         jmp   start
C999:                   266 *
C999:                   267 * Whew! Now send the damn checksum as two FM bytes
C999:                   268 *
C999:         C999      269 datdone equ   *
C999:A5 40        (3)   270         lda   checksum      ;c7 c6 c5 c4 c3 c2 c1 c0
C99B:09 AA        (2)   271         ora   #$AA          ; 1 c6  1 c4  1 c2  1 c0
C99D:BC 8C C0     (4)   272 scm1    ldy   l6clr,x
C9A0:10 FB   C99D(3)    273         bpl   scm1          ;Handshake this byte
C9A2:9D 8D C0     (5)   274         sta   l6set,x       ;These are even bits
C9A5:                   275 *
C9A5:A5 40        (3)   276         lda   checksum      ;c7 c6 c5 c4 c3 c2 c1 c0
C9A7:4A           (2)   277         lsr   a             ; 0 c7 c6 c5 c4 c3 c2 c1
C9A8:09 AA        (2)   278         ora   #$AA          ; 1 c7  1 c5  1 c3  1 c1
C9AA:20 53 CA     (6)   279         jsr   sendbyte
C9AD:                   280 *
C9AD:                   281 * Send the end of packet mark
C9AD:                   282 *
C9AD:A9 C8        (2)   283         lda   #packetend
C9AF:20 53 CA     (6)   284         jsr   sendbyte
C9B2:                   285 *
C9B2:                   286 * Wait until write underflow
C9B2:                   287 *
C9B2:BD 8C C0     (4)   288 sd7     lda   l6clr,x
C9B5:29 40        (2)   289         and   #$40
C9B7:D0 F9   C9B2(3)    290         bne   sd7           ;Still writing data
C9B9:                   291 *
C9B9:9D 8D C0     (5)   292         sta   l6set,x       ;Back to sense mode (dummy write)
C9BC:                   293 *
```

```
C9BC:                    294 * Now wait until the drive acknowledges reciept of the
C9BC:                    295 *  string or until timeout
C9BC:                    296 *
C9BC:A0 0A       (2)     297         ldy   #bsyto2       ;Load timeout to see bsy low
C9BE:88          (2)     298 patch1 dey                   ;A little closer to an error
C9BF:D0 08   C9C9(3)     299         bne   sd9           ;There's still time
C9C1:                    300 *
C9C1:                    301 * Too much time has elapsed.  Drive didn't get string.
C9C1:                    302 *
C9C1:A9 01       (2)     303         lda   #noanswer     ;Report error in comm error byte
C9C3:        C9C3        304 dberror equ  *
C9C3:20 9A CA    (6)     305         jsr   SetXN0        ;For dberror entry
C9C6:38          (2)     306         sec                 ;Signal a problem
C9C7:B0 06   C9CF(3)     307         bcs   sd10
C9C9:                    308 *
C9C9:                    309 * See if drive has acknowledged the bytes yet
C9C9:                    310 *
C9C9:BD 8E C0    (4)     311 sd9     lda   l7clr,x       ;Wait 'til /BSY lo
C9CC:30 F0   C9BE(3)     312         bmi   patch1
C9CE:                    313 *
C9CE:                    314 * Finish the sequence
C9CE:                    315 *
C9CE:18          (2)     316         clc                 ;This is a normal exit
C9CF:BD 80 C0    (4)     317 sd10    lda   reqclr,x      ;Set REQ lo
C9D2:BD 8C C0    (4)     318         lda   l6clr,x       ;Back into read mode
C9D5:                    319 *
C9D5:                    320 * Pull back the bytecount in all cases
C9D5:                    321 *
C9D5:60          (6)     322         rts
C9D6:                    323 *
C9D6:                    324 *
C9D6:                    325 * This table, when sent in reverse order, provides a
C9D6:                    326 *  sync pattern used to synchronize the drive IWM with
C9D6:                    327 *  the data stream.  The first byte (last sent) is the
C9D6:                    328 *  packet begin mark.
C9D6:                    329 *
C9D6:C3                  330 preamble dfb packetbeg
C9D7:FF FC F3 CF         331 synctab dfb  $FF,$FC,$F3,$CF,$3F
C9DC:                    332 *
C9DC:                    333 *
```

```
C9DC:                   335 *
C9DC:                   336 * These routines are for wasting specific amounts of time
C9DC:                   337 *  This code segment should not cross page boundaries.
C9DC:                   338 *
C9DC:20 E1 C9     (6)   339 waste32 jsr  waste14
C9DF:EA           (2)   340 waste18 nop
C9E0:EA           (2)   341 waste16 nop
C9E1:EA           (2)   342 waste14 nop
C9E2:60           (6)   343 waste12 rts
C9E3:                   344 *
C9E3:                   345 *
C9E3:         C9E3      346 markerr equ  *
C9E3:4C C3 C9     (3)   347        jmp   dberror
```

```
C9E6:                  349 ********************************************************
C9E6:                  350 *                                                      *
C9E6:                  351 *  ReceivePack       Get a packet from bus resident    *
C9E6:                  352 *                                                      *
C9E6:                  353 *                  ____________________________        *
C9E6:                  354 *  REQ _______|2                          5|_________  *
C9E6:                  355 *              ____________________________            *
C9E6:                  356 * /BSY ___|1              3             4|__________  *
C9E6:                  357 *                                                      *
C9E6:                  358 *    1)  Drive signals ready to send packet            *
C9E6:                  359 *    2)  Host signals ready to recieve data            *
C9E6:                  360 *    3)  Packet is transmitted (sync, mark, IDs, data, *
C9E6:                  361 *         checksum [msb=1])                            *
C9E6:                  362 *    4)  Drive signals packet dispatched               *
C9E6:                  363 *    5)  Host acknowledges reciept of packet           *
C9E6:                  364 *                                                      *
C9E6:                  365 *    The bytes are sent in slow mode (32 cycles/byte)  *
C9E6:                  366 *  and the timing is critical.  Branches which should  *
C9E6:                  367 *  not cross page boundaries are marked.               *
C9E6:                  368 *                                                      *
C9E6:                  369 *   Input:  buffer <- address where packet guts left   *
C9E6:                  370 *                                                      *
C9E6:                  371 *   Output: carry set- handshake error                 *
C9E6:                  372 *                   clr- bytes recieved                *
C9E6.                  373 *            A <- error0 if carry set                  *
C9E6:                  374 *                                                      *
C9E6:                  375 ********************************************************
C9E6:                  376 *
C9E6:         C9E6     377 grabstatus equ *
C9E6:         C9E6     378 ReceivePack equ *
C9E6:                  379 *
C9E6:                  380 * Init the checksum
C9E6:                  381 *
C9E6:A9 00       (2)   382         lda   #$00
C9E8:85 40       (3)   383         sta   checksum
C9EA:                  384 *
C9EA:                  385 * Copy over buffer -> buffer2
C9EA:                  386 *
C9EA:A5 54       (3)   387         lda   buffer
C9EC:85 56       (3)   388         sta   buffer2
C9EE:A5 55       (3)   389         lda   buffer+1
C9F0:85 57       (3)   390         sta   buffer2+1
C9F2:                  391 *
C9F2:                  392 * Set up the indirect pointer for jump to 2nd part of code
C9F2:                  393 *
C9F2:         0001     394         ifeq  IIc^ROM       ;Don't do in //c version
C9F2:                  401         fin
C9F2:                  402 *
C9F2:20 80 CA    (6)   403         jsr   enablechain   ;Set X register to $N0
C9F5:                  404 *
C9F5:BD 8D C0    (4)   405         lda   l6set,x       ;Prep for sense mode
C9F8:                  406 *
C9F8:                  407 * Now wait for BSY to go hi, signalling 'ready w/ status'
C9F8:                  408 *
C9F8:BD 8E C0    (4)   409 rdh1    lda   l7clr,x       ;Read sense
C9FB:10 FB   C9F8(3)   410         bpl   rdh1          ;Wait til a high
C9FD:                  411 *
C9FD:                  412 * Signal Liron we're ready to recieve
```

```
C9FD:                   413 *
C9FD:BD 81 C0     (4)   414         lda   reqset,x      ;Raise /REQ
CA00:                   415 *
CA00:                   416 * Wait for a byte from Liron or timeout
CA00:                   417 *
CA00:A0 1E        (2)   418         ldy   #statmto      ;Max bytes 'til stat mark
CA02:BD 8C C0     (4)   419 rdh2    lda   16clr,x
CA05:10 FB    CA02(3)   420         bpl   rdh2          ;*** No Page Cross ***
CA07:88           (2)   421         dey
CA08:30 D9    C9E3(3)   422         bmi   markerr       ;Didn't find a packet in time
CA0A:                   423 *
CA0A:                   424 * Is it the beginning of the packet?
CA0A:                   425 *
CA0A:C9 C3        (2)   426         cmp   #packetbeg    ;Find the packet begin mark
CA0C:D0 F4    CA02(3)   427         bne   rdh2          ;Back again - no timeout for now
CA0E:                   428 *
CA0E:                   429 * Okay load up the table with this stuff
CA0E:                   430 *
CA0E:         CA0E      431 rdh5    equ   *
CA0E:                   432 *
CA0E:A0 06        (2)   433         ldy   #6            ;Seven bytes of overhead
CA10:BD 8C C0     (4)   434 rdh3    lda   16clr,x       ;If byte ready, grab it
CA13:10 FB    CA10(3)   435         bpl   rdh3          ;*** No Page Cross ***
CA15:29 7F        (2)   436         and   #%01111111    ;Strip start bit
CA17:99 4B 00     (5)   437         sta   rcvbuf,y
CA1A:49 80        (2)   438         eor   #$80          ;Pop MSB back on for checksum
CA1C:45 40        (3)   439         eor   checksum
CA1E:85 40        (3)   440         sta   checksum
CA20:88           (2)   441         dey
CA21:10 ED    CA10(3)   442         bpl   rdh3
CA23:                   443 *
CA23:                   444 * Set groups of seven buffer pointer buffer2
CA23:                   445 *
CA23:A5 4C        (3)   446         lda   oddbytes
CA25:F0 27    CA4E(3)   447         beq   start2        ;Skip alteration if no oddbytes
CA27:18           (2)   448         clc
CA28:65 54        (3)   449         adc   buffer
CA2A:85 56        (3)   450         sta   buffer2
CA2C:A5 55        (3)   451         lda   buffer+1
CA2E:69 00        (2)   452         adc   #0
CA30:85 57        (3)   453         sta   buffer2+1
CA32:                   454 *
CA32:A0 00        (2)   455         ldy   #0
CA34:                   456 *
CA34:                   457 * Now receive the odd bytes
CA34:                   458 *
CA34:BD 8C C0     (4)   459 start0  lda   16clr,x       ;Read in the odd bytes topbits
CA37:10 FB    CA34(3)   460         bpl   start0
CA39:0A           (2)   461         asl   a             ;Pop off the start bit
CA3A:85 41        (3)   462         sta   topbits
CA3C:         CA3C      463 start1  equ   *
CA3C:BD 8C C0     (4)   464         lda   16clr,x       ;Get an odd byte
CA3F:10 FB    CA3C(3)   465         bpl   start1
CA41:06 41        (5)   466         asl   topbits       ;Get an MSB
CA43:B0 02    CA47(3)   467         bcs   gob1          ;If MSB set, leave start bit
CA45:49 80        (2)   468         eor   #$80          ;MSB clear- flip start bit
CA47:91 54        (6)   469 gob1    sta   (buffer),y    ;Squirrel it away
CA49:C8           (2)   470         iny                 ;Next spot
```

```
CA4A:C4 4C       (3)  471         cpy   oddbytes      ;Are we done?
CA4C:90 EE   CA3C(3)  472         blt   start1        ;If more, branch
CA4E:                 473 *
CA4E:        CA4E     474 start2 equ   *
CA4E:        0001     475         do    IIc^ROM
CA4E:4C 72 CC    (3)  476         jmp   SlotDepRd
CA51:                 477         else
CA51:                 479         fin
CA51:                 480 *
CA51:        CA51     481 Send80 equ   *
CA51:A9 80       (2)  482         lda   #$80
CA53:        CA53     483 SendByte equ *
CA53:BC 8C C0    (4)  484         ldy   l6clr,x
CA56:10 FB   CA53(3)  485         bpl   SendByte
CA58:9D 8D C0    (5)  486         sta   l6set,x
CA5B:45 40       (3)  487         eor   checksum
CA5D:85 40       (3)  488         sta   checksum
CA5F:60          (6)  489         rts
CA60:                 490 *
CA60:                 491 *
CA60:                 492 *
CA60:                 493 *
CA60:        CA60     494 resetchain equ *
CA60:20 8A CA    (6)  495         jsr   ClrPhases
CA63:BD 81 C0    (4)  496         lda   reqset,x
CA66:BD 85 C0    (4)  497         lda   ca2set,x
CA69:A0 50       (2)  498         ldy   #80             ;Hard reset for 80 ms
CA6B:20 73 CA    (6)  499         jsr   YMSWait
CA6E:                 500 *
CA6E:20 8A CA    (6)  501         jsr   ClrPhases
CA71:                 502 *
CA71:A0 0A       (2)  503         ldy   #10             ;About 10 mS reset time!
CA73:                 504 *
CA73:        CA73     505 YMSWait equ  *
CA73:20 7A CA    (6)  506         jsr   OneMS
CA76:88          (2)  507         dey
CA77:D0 FA   CA73(3)  508         bne   YMSWait
CA79:60          (6)  509         rts
CA7A:                 510 *
CA7A:        CA7A     511 OneMS  equ   *
CA7A:A2 C8       (2)  512         ldx   #200
CA7C:CA          (2)  513 onems1 dex
CA7D:D0 FD   CA7C(3)  514         bne   onems1
CA7F:60          (6)  515         rts
CA80:                 516 *
CA80:                 517 *
CA80:        CA80     518 enablechain equ *
CA80:20 9A CA    (6)  519         jsr   SetXN0
CA83:BD 83 C0    (4)  520         lda   ca1set,x
CA86:BD 87 C0    (4)  521         lda   lstrbset,x
CA89:60          (6)  522         rts
CA8A:                 523 *
CA8A:                 524 *
CA8A:        CA8A     525 ClrPhases equ *
CA8A:20 9A CA    (6)  526         jsr   SetXN0
CA8D:BD 80 C0    (4)  527         lda   reqclr,x
CA90:BD 82 C0    (4)  528         lda   ca1clr,x
CA93:BD 84 C0    (4)  529         lda   ca2clr,x
```

```
CA96:BD 86 C0    (4)  530         lda   lstrbclr,x
CA99:60          (6)  531         rts
CA9A:                 532 *
CA9A:                 533 *
CA9A:         CA9A    534 SetXN0 equ   *
CA9A:         0001    535         do    IIc
CA9A:A2 60       (2)  536         ldx   #$60
CA9C:                 537         else
CA9C:                 544         fin
CA9C:                 545 *
CA9C:60          (6)  546         rts
CA9D:                 547 *
CA9D:                 548 * Shift tables for use when reading.  Each table should not
CA9D:                 549 *  straddle pages.
CA9D:                 550 *
CA9D:80 80 80 80      551 shift1 dfb   $80,$80,$80,$80,$80,$80,$80,$80
CAA5:00 00 00 00      552         dfb   0,0,0,0,0,0,0,0
CAAD:80 80 80 80      553 shift2 dfb   $80,$80,$80,$80,0,0,0,0
CAB5:80 80 80 80      554         dfb   $80,$80,$80,$80,0,0,0,0
CABD:80 80 00 00      555 shift3 dfb   $80,$80,0,0,$80,$80,0,0
CAC5:80 80 00 00      556         dfb   $80,$80,0,0,$80,$80,0,0
CACD:80 00 80 00      557 shift4 dfb   $80,0,$80,0,$80,0,$80,0
CAD5:80 00 80 00      558         dfb   $80,0,$80,0,$80,0,$80,0
CADD:                 559 *
CADD:                 560 *
```

```
CADD:                   562 *
CADD:           CADD    563 SendData equ *
CADD:A9 05        (2)   564         lda   #>RC2
CADF:A0 00        (2)   565         ldy   #<RC2
CAE1:20 00 CB     (6)   566         jsr   SendPile
CAE4:90 05   CAEB(3)    567         bcc   sdoubt
CAE6:A9 80        (2)   568         lda   #CommReset
CAE8:20 90 CF     (6)   569         jsr   AssignID
CAEB:           CAEB    570 sdoubt equ   *
CAEB:60           (6)   571         rts
CAEC:                   572 *
CAEC:                   573 *
CAEC:           CAEC    574 SendPack equ *
CAEC:20 00 CB     (6)   575         jsr   SendPile      ;Try to send a pack
CAEF:90 FA   CAEB(3)    576         bcc   sdoubt
CAF1:A9 80        (2)   577         lda   #CommReset    ;This is a communications failure
CAF3:20 90 CF     (6)   578         jsr   AssignID      ;Reset to try again
CAF6:                   579 *
CAF6:AD F8 06     (4)   580         lda   SvBcL         ;Get back the packetlength
CAF9:85 4D        (3)   581         sta   bytecountl
CAFB:AD 78 07     (4)   582         lda   SvBcH
CAFE:85 4E        (3)   583         sta   bytecounth
CB00:                   584 *
CB00:           CB00    585 SendPile equ *
CB00:A9 B8        (2)   586         lda   #>RC1         ;Retry count (big!)
CB02:A0 0B        (2)   587         ldy   #<RC1
CB04:                   588 *
CB04:           CB04    589 AltSendPile equ *
CB04:A6 58        (3)   590         ldx   slot
CB06:9D F3 04     (5)   591         sta   Retry,x
CB09:98           (2)   592         tya
CB0A:9D 73 05     (5)   593         sta   Retry2,x
CB0D:                   594 *
CB0D:                   595 * SendPack destroys the bytecount
CB0D:                   596 *
CB0D:           CB0D    597 spile1 equ   *
CB0D:A5 4D        (3)   598         lda   bytecountl
CB0F:8D F8 06     (4)   599         sta   SvBcL
CB12:A5 4E        (3)   600         lda   bytecounth
CB14:8D 78 07     (4)   601         sta   SvBcH
CB17:                   602 *
CB17:20 86 C8     (6)   603         jsr   SendOnePack   ;Send the packet
CB1A:                   604 *
CB1A:AD F8 06     (4)   605         lda   SvBcL
CB1D:85 4D        (3)   606         sta   bytecountl
CB1F:AD 78 07     (4)   607         lda   SvBcH
CB22:85 4E        (3)   608         sta   bytecounth
CB24:                   609 *
CB24:90 0C   CB32(3)    610         bcc   spilout
CB26:A6 58        (3)   611         ldx   slot
CB28:DE F3 04     (7)   612         dec   Retry,x
CB2B:D0 E0   CB0D(3)    613         bne   spile1
CB2D:DE 73 05     (7)   614         dec   Retry2,x
CB30:10 DB   CB0D(3)    615         bpl   spile1       ;If all fails, carry is set
CB32:60           (6)   616 spilout rts
CB33:                   617 *
CB33:           CB33    618 RecPack equ  *
CB33:A4 58        (3)   619         ldy   Slot
```

```
CB35:A9 05      (2)  620         lda   #>RC2
CB37:99 F3 04   (5)  621         sta   Retry,y
CB3A:       CB3A     622 rpk1    equ   *
CB3A:20 E6 C9   (6)  623         jsr   ReceivePack
CB3D:90 0F   CB4E(3) 624         bcc   rpout
CB3F:A0 01      (2)  625         ldy   #1
CB41:20 73 CA   (6)  626         jsr   YMSWait
CB44:20 C3 C9   (6)  627         jsr   dberror       ;Recycle handshake and set carry
CB47:A6 58      (3)  628         ldx   Slot
CB49:DE F3 04   (7)  629         dec   Retry,x
CB4C:D0 EC   CB3A(3) 630         bne   rpk1          ;Carry set still
CB4E:       CB4E     631 rpout   equ   *
CB4E:60         (6)  632         rts
CB4F:                633 *
CB4F:                634 *
```

```
CB4F:                  636 ************************************************************
CB4F:                  637 *                                                          *
CB4F:                  638 *  Divide7                Do DIV and MOD 7 and set auxptr  *
CB4F:                  639 *                                                          *
CB4F:                  640 *   This routine divides the bytecount by seven.  The      *
CB4F:                  641 *  quotient gives the number of groups of seven bytes to   *
CB4F:                  642 *  be sent, and the remainder gives the number of "odd"    *
CB4F:                  643 *  bytes.                                                  *
CB4F:                  644 *                                                          *
CB4F:                  645 *  Input:   bytecount1,h <- # of bytes to write            *
CB4F:                  646 *           buffer       <- pointer to data                *
CB4F:                  647 *  Output:  auxptr       <- pointer to speed up csumming   *
CB4F:                  648 *           oddbytes     <- bytecount MOD 7                *
CB4F:                  649 *           grp7ctr      <- bytecount DIV 7                *
CB4F:                  650 *                                                          *
CB4F:                  651 ************************************************************
CB4F:                  652 *
CB4F:00 24 49          653 pdiv7tab dfb 0,36,73
CB52:00 04 01          654 pmod7tab dfb 0,4,1
CB55:00 01 02 04       655 div7tab dfb  0,1,2,4,9,18
CB5B:00 01 02 04       656 mod7tab dfb  0,1,2,4,1,2
CB61:                  657 *
CB61:00 7F FF          658 auxptrinc dfb 0,$7F,$FF
CB64:                  659 *
CB64:         CB64     660 WritePrep equ *
CB64:         CB64     661 Divide7 equ  *
CB64:                  662 *
CB64:                  663 * Set up auxptr <- buffer+$80   if $0FF < bytecount < $200
CB64:                  664 *     or auxptr <- buffer+$100  if $1FF < bytecount
CB64:                  665 *
CB64:A6 4E       (3)   666         ldx   bytecounth     ;0, 1 or 2
CB66:F0 13   CB7B(3)   667         beq   noauxptr       ;Auxptr used only for full pages
CB68:                  668 *
CB68:A5 55       (3)   669         lda   buffer+1
CB6A:85 57       (3)   670         sta   auxptr+1       ;Copy over hi order part
CB6C:                  671 *
CB6C:A9 80       (2)   672         lda   #$80           ;Anticipate smaller bytecount
CB6E:E0 01       (2)   673         cpx   #1             ;Check bytecount
CB70:F0 04   CB76(3)   674         beq   sap1           ;=> $0FF < bytecount < $200
CB72:                  675 *
CB72:E6 57       (5)   676         inc   auxptr+1       ;Add $100 to bytecount instead
CB74:A9 00       (2)   677         lda   #0             ;Make sure lo order unaltered
CB76:18          (2)   678 sap1    clc
CB77:65 54       (3)   679         adc   buffer
CB79:85 56       (3)   680         sta   auxptr
CB7B:                  681 *
CB7B:                  682 * Now look up the first order guess for DIV and MOD.  X still has
CB7B:                  683 *  bytecount DIV 256.
CB7B:                  684 *
CB7B:         CB7B     685 noauxptr equ *
CB7B:BD 4F CB    (4)   686         lda   pdiv7tab,x
CB7E:85 4B       (3)   687         sta   grp7ctr
CB80:BD 52 CB    (4)   688         lda   pmod7tab,x
CB83:85 4C       (3)   689         sta   oddbytes
CB85:                  690 *
CB85:                  691 * Now add in the mods and divs for each of the five hi order
CB85:                  692 *  bits in the lo order bytecount, correcting each time MOD becomes
CB85:                  693 *  bigger than 6.
```

```
CB85:                   694 *
CB85:A2 05        (2)   695         ldx   #5              ;Do for five bits
CB87:A5 4D        (3)   696         lda   bytecount1
CB89:85 59        (3)   697         sta   temp            ;Store lo order for shifting
CB8B:29 07        (2)   698         and   #%00000111      ;Save lo three for later
CB8D:A8           (2)   699         tay
CB8E:                   700 *
CB8E:         CB8E      701 divide3 equ   *
CB8E:06 59        (5)   702         asl   temp            ;C <- next from bytecount1
CB90:90 15    CBA7(3)   703         bcc   divide2         ;If clear, no effect on DIV,MOD
CB92:BD 5B CB     (4)   704         lda   mod7tab,x       ;Get MOD7 for 2^n
CB95:         CB95      705 divide4 equ   *
CB95:18           (2)   706         clc
CB96:65 4C        (3)   707         adc   oddbytes        ;Got new MOD value
CB98:C9 07        (2)   708         cmp   #7              ;Is it too big?
CB9A:90 02    CB9E(3)   709         blt   divide1         ;=> NO leave MOD - 0->C
CB9C:E9 07        (2)   710         sbc   #7              ;Bring MOD under 7 - C still set
CB9E:         CB9E      711 divide1 equ   *
CB9E:85 4C        (3)   712         sta   oddbytes
CBA0:BD 55 CB     (4)   713         lda   div7tab,x       ;Get DIV for this 2^n
CBA3:65 4B        (3)   714         adc   grp7ctr         ;Add to DIV along with correction (C)
CBA5:85 4B        (3)   715         sta   grp7ctr         ;Update the DIV
CBA7:         CBA7      716 divide2 equ   *
CBA7:CA           (2)   717         dex                   ;One less bit to deal with
CBA8:30 06    CBB0(3)   718         bmi   divide5         ;Escape after 6 times through loop
CBAA:D0 E2    CB8E(3)   719         bne   divide3         ;Take brnch 1st 5 loops
CBAC:                   720 *
CBAC:98           (2)   721         tya                   ;Get back the last three bits
CBAD:4C 95 CB     (3)   722         jmp   divide4         ;Sixth pass add in remains
CBB0:                   723 *
CBB0:         CBB0      724 divide5 equ   *
CBB0:                   725 *
CBB0:                   726 *
```

```
CBB0:                  728 ************************************************************
CBB0:                  729 *                                                          *
CBB0:                  730 *  PreCheck                    Does the checksumming prepass *
CBB0:                  731 *                                                          *
CBB0:                  732 *    Input:   bytecount   <- bytes in buffer               *
CBB0:                  733 *             buffer      <- pointer to data to send       *
CBB0:                  734 *             auxptr      <- extra pointer to speed process*
CBB0:                  735 *    Output:  checksum    <- 8 bit XOR of data to be sent  *
CBB0:                  736 *                                                          *
CBB0:                  737 ************************************************************
CBB0:                  738 *
CBB0:          CBB0    739 PreCheck equ *
CBB0:                  740 *
CBB0:                  741 * Checksum any full pages
CBB0:                  742 *
CBB0:A5 55        (3)  743         lda   buffer+1
CBB2:48           (3)  744         pha                   ;Preserve buffer pointer
CBB3:A9 00        (2)  745         lda   #0
CBB5:A6 4E        (3)  746         ldx   bytecounth
CBB7:F0 16   CBCF(3)   747         beq   lastpass        ;If no complete pages, skip this
CBB9:          CBB9    748 xor2    equ   *
CBB9:BC 61 CB     (4)  749         ldy   auxptrinc,x     ;Get number of bytes each ptr
CBBC:          CBBC    750 xor1    equ   *
CBBC:51 54        (5)  751         eor   (buffer),y
CBBE:51 56        (5)  752         eor   (auxptr),y
CBC0:88           (2)  753         dey                   ;One less
CBC1:D0 F9   CBBC(3)   754         bne   xor1
CBC3:51 54        (5)  755         eor   (buffer),y
CBC5:51 56        (5)  756         eor   (auxptr),y      ;Have to deal with 0 case
CBC7:                  757 *
CBC7:                  758 * Now move the buffer up for next section
CBC7:                  759 *
CBC7:E0 01        (2)  760         cpx   #1
CBC9:F0 02   CBCD(3)   761         beq   xor5            ;If 256 and up bytes, bump x1
CBCB:E6 55        (5)  762         inc   buffer+1        ; otherwise x2
CBCD:E6 55        (5)  763 xor5    inc   buffer+1
CBCF:                  764 *
CBCF:          CBCF    765 lastpass equ *
CBCF:                  766 *
CBCF:                  767 * Do the remaining less than a page with a single pointer
CBCF:                  768 *
CBCF:A4 4D        (3)  769         ldy   bytecount
CBD1:F0 09   CBDC(3)   770         beq   xor4
CBD3:51 54        (5)  771         eor   (buffer),y      ;Compensate for nth byte
CBD5:51 54        (5)  772 xor3    eor   (buffer),y
CBD7:88           (2)  773         dey
CBD8:D0 FB   CBD5(3)   774         bne   xor3
CBDA:51 54        (5)  775         eor   (buffer),y      ;Last damn (0th) byte
CBDC:                  776 *
CBDC:                  777 * Store result away.  Retrieve old buffer value.
CBDC:                  778 *
CBDC:          CBDC    779 xor4    equ   *
CBDC:85 40        (3)  780         sta   checksum
CBDE:68           (4)  781         pla
CBDF:85 55        (3)  782         sta   buffer+1
CBE1:                  783 *
CBE1:                  784 *
```

```
CBE1:                  786 ********************************************************
CBE1:                  787 *                                                      *
CBE1:                  788 * DetTopBits              Get topbits for odd bytes   *
CBE1:                  789 *                                                      *
CBE1:                  790 *   Also sets buffer2 pointer to pointer at groups of *
CBE1:                  791 *  seven bytes.                                        *
CBE1:                  792 *                                                      *
CBE1:                  793 *  Input:   oddbytes <- # of "odd" bytes               *
CBE1:                  794 *           buffer   <- pointer to data                *
CBE1:                  795 *  Output:  tbodd    <- topbits for odd bytes          *
CBE1:                  796 *           buffer2  <- buffer+oddbytes                 *
CBE1:                  797 *                                                      *
CBE1:                  798 ********************************************************
CBE1:                  799 *
CBE1:          CBE1    800 DetTopBits equ *
CBE1:                  801 *
CBE1:A4 4C       (3)   802         ldy   oddbytes
CBE3:88          (2)   803         dey
CBE4:A9 00       (2)   804         lda   #0
CBE6:85 59       (3)   805         sta   tbodd
CBE8:                  806 *
CBE8:B1 54       (5)   807 gtbob   lda   (buffer),y
CBEA:0A          (2)   808         asl   a
CBEB:66 59       (5)   809         ror   tbodd
CBED:88          (2)   810         dey
CBEE:10 F8   CBE8(3)   811         bpl   gtbob
CBF0:38          (2)   812         sec
CBF1:66 59       (5)   813         ror   tbodd
CBF3:                  814 *
CBF3:A5 4C       (3)   815         lda   oddbytes
CBF5:18          (2)   816         clc
CBF6:65 54       (3)   817         adc   buffer
CBF8:85 56       (3)   818         sta   buffer2
CBFA:A5 55       (3)   819         lda   buffer+1
CBFC:69 00       (2)   820         adc   #0
CBFE:85 57       (3)   821         sta   buffer2+1
CC00:                  822 *
CC00:                  823 *
```

```
CC00:                  825 ************************************************************
CC00:                  826 *                                                          *
CC00:                  827 *  Sun                     Set up next buffer and topbits  *
CC00:                  828 *                                                          *
CC00:                  829 *   Primes the pipe for the group of seven bytes routine *
CC00:                  830 *  setting the topbits byte and the "next" buffer.         *
CC00:                  831 *  The routine also advances the buffer pointer by 7 to    *
CC00:                  832 *  prepare for the groups of seven transfer.               *
CC00:                  833 *                                                          *
CC00:                  834 *    Input:   buffer2   <- points to groups of 7 data      *
CC00:                  835 *    Output:  next1,7   <- first 7 bytes in buffer         *
CC00:                  836 *             topbits   <- MSBs of first 7 bytes           *
CC00:                  837 *                                                          *
CC00:                  838 ************************************************************
CC00:                  839 *
CC00:          CC00    840 Sun    equ   *
CC00:                  841 *
CC00:                  842 * Copy first seven bytes into the pipeline
CC00:                  843 *
CC00:A0 06       (2)   844        ldy   #6
CC02:38          (2)   845 sun2   sec
CC03:B1 56       (5)   846        lda   (buffer2),y
CC05:99 4D 00    (5)   847        sta   next,y
CC08:30 01   CC0B(3)   848        bmi   sun1
CC0A:18          (2)   849        clc
CC0B:66 41       (5)   850 sun1   ror   topbits
CC0D:88          (2)   851        dey
CC0E:10 F2   CC02(3)   852        bpl   sun2
CC10:38          (2)   853        sec
CC11:66 41       (5)   854        ror   topbits
CC13:                  855 *
CC13:                  856 * Advance the pointer
CC13:                  857 *
CC13:A5 56       (3)   858        lda   buffer2
CC15:18          (2)   859        clc
CC16:69 07       (2)   860        adc   #7
CC18:85 56       (3)   861        sta   buffer2
CC1A:90 02   CC1E(3)   862        bcc   sun3
CC1C:E6 57       (5)   863        inc   buffer2+1
CC1E:          CC1E    864 sun3   equ   *
CC1E:60          (6)   865        rts
CC1F:                  866 *
CC1F:                  867 *
```

```
CC1F:                   869 *
CC1F:                   870 * X is slot*16, Y is the desired mode
CC1F:                   871 *
CC1F:                   872 * Set up the IWM mode register.  Extreme care should be taken
CC1F:                   873 *  here. Setting the mode byte with indexed stores causes a false
CC1F:                   874 *  byte to be written a cycle before the real value is written.
CC1F:                   875 *  This false value, if it enables the timer, causes the IWM Rev A
                            to
CC1F:                   876 *  pop the motor on, inhibiting the setting of the mode until the
CC1F:                   877 *  motor times out! We avoid this by setting the mode byte only
                            when
CC1F:                   878 *  it is not what we want, and if it's not we stay here until we
CC1F:                   879 *  see that it is what we want.
CC1F:                   880 *
CC1F:         CC1F      881 SetIWMode equ *
CC1F:BD 88 C0    (4)    882         lda   monclr,x      ;Motor must be off
CC22:BD 8D C0    (4)    883         lda   l6set,x       ;Set up to access mode register
CC25:4C 2C CC    (3)    884         jmp   careful       ;Don't mess unless we gotta
CC28:98          (2)    885 biz     tya
CC29:9D 8F C0    (5)    886         sta   l7set,x       ;Try storing the mode value
CC2C:         CC2C      887 careful equ   *
CC2C:98          (2)    888         tya                 ;Get back the target value
CC2D:5D 8E C0    (4)    889         eor   l7clr,x       ;Compare with observed value
CC30:29 1F       (2)    890         and   #$1F          ;Can only read low 5 bits
CC32:D0 F4    CC28(3)   891         bne   biz           ;If not right, back to try again
CC34:60          (6)    892         rts
CC35:                   893 *
CC35:                   894 *
CC35:         0001      895         Do    IIc
CC35:         CC35      896 WaitIWMOff equ *
CC35:                   897 *
CC35:                   898 * Make sure you're in read mode and wait 'til Disk // motor is off
CC35:                   899 *
CC35:20 9A CA    (6)    900         jsr   SetXN0        ;Set X
CC38:BD 8E C0    (4)    901         lda   l7clr,x
CC3B:BD 8D C0    (4)    902         lda   l6set,x
CC3E:         CC3E      903 wiwm1   equ   *
CC3E:BD 8E C0    (4)    904         lda   l7clr,x
CC41:29 20       (2)    905         and   #%00100000
CC43:D0 F9    CC3E(3)   906         bne   wiwm1
CC45:BD 8C C0    (4)    907         lda   l6clr,x
CC48:                   908 *
CC48:                   909 * Wait an additional 700 microseconds to allow 12V on Disk // to
                            decay
CC48:                   910 *
CC48:5A          (3)    911         phy
CC49:A0 8C       (2)    912         ldy   #140
CC4B:88          (2)    913 wiwm2   dey
CC4C:D0 FD    CC4B(3)   914         bne   wiwm2
CC4E:7A          (4)    915         ply
CC4F:                   916 *
CC4F:60          (6)    917         rts
CC50:                   918         fin
CC50:                   919 *
CC50:                   920 *
CC50:                   921 * This takes grp7ctr and oddbytes and calculates
                            7*grp7ctr+oddbytes.
CC50:                   922 *  The results are in Y(hi) and A(lo).  This is the number of bytes
CC50:                   923 *  that were received in the last ReceivePack.
CC50:                   924 *
CC50:         CC50      925 Rcvcount equ *
CC50:A5 4B       (3)    926         lda   grp7ctr
```

```
CC52:A8          (2)  927         tay
CC53:A2 00       (2)  928         ldx   #0
CC55:86 4B       (3)  929         stx   grp7ctr
CC57:A2 03       (2)  930         ldx   #3
CC59:0A          (2)  931 times7  asl   a
CC5A:26 4B       (5)  932         rol   grp7ctr
CC5C:CA          (2)  933         dex
CC5D:D0 FA    CC59(3) 934         bne   times7
CC5F:18          (2)  935         clc
CC60:65 4C       (3)  936         adc   oddbytes
CC62:90 02    CC66(3) 937         bcc   t71
CC64:E6 4B       (5)  938         inc   grp7ctr
CC66:84 4C       (3)  939 t71     sty   oddbytes
CC68:38          (2)  940         sec
CC69:E5 4C       (3)  941         sbc   oddbytes
CC6B:B0 02    CC6F(3) 942         bcs   t72
CC6D:C6 4B       (5)  943         dec   grp7ctr
CC6F:A4 4B       (3)  944 T72     ldy   grp7ctr
CC71:60          (6)  945         rts
CC72:                 946 *
CC72:                 947 *
CC72:                 136 *
CC72:         0001    137         do    IIc^ROM
CC72:                 138         include pc.read
CC72:         CC72      1 SlotDepRd equ *
CC72:         CC72      2 start25 equ  *
CC72:A0 00       (2)    3         ldy   #0
CC74:A5 4B       (3)    4         lda   grp7ctr
CC76:48          (3)    5         pha                    ;Save groups of seven counter
CC77:D0 03    CC7C(3)   6         bne   start35
CC79:4C 09 CD    (3)    7         jmp   done5            ;Go get the checksum
CC7C:                   8 *
CC7C:                   9 * Okay, get the groups of seven
CC7C:                  10 *  Start by getting the topbits for this group of seven
CC7C:                  11 *
CC7C:         CC7C     12 start35 equ  *
CC7C:AD EC C0    (4)   13         lda   l6clr+TheOff     ;Get topbits
CC7F:10 FB    CC7C(3)  14         bpl   start35
CC81:85 59       (3)   15         sta   temp             ;Just a second
CC83:                  16 *
CC83:                  17 * Split up the seven bits into two indices for topbit tables
CC83:                  18 *
CC83:4A          (2)   19         lsr   a                ;0   1  d1  d2  d3  d4  d5  d6
CC84:4A          (2)   20         lsr   a                ;0   0   1  d1  d2  d3  d4  d5
CC85:4A          (2)   21         lsr   a                ;0   0   0   1  d1  d2  d3  d4
CC86:29 0F       (2)   22         and   #%00001111       ;0   0   0   0  d1  d2  d3  d4
CC88:AA          (2)   23         tax                    ;First index into the tables
CC89:A5 59       (3)   24         lda   temp             ;1  d1  d2  d3  d4  d5  d6  d7
CC8B:29 07       (2)   25         and   #%00000111       ;0   0   0   0   0  d5  d6  d7
CC8D:85 59       (3)   26         sta   temp             ;Keep for last three bytes
CC8F:                  27 *
CC8F:                  28 * Now read the first byte, reunite its msb, store it, and checksum
                          it.
CC8F:                  29 *
CC8F:AD EC C0    (4)   30         lda   l6clr+TheOff
CC92:10 FB    CC8F(3)  31         bpl   *-3              ;Back 1 instruction
CC94:5D 9D CA    (4)   32         eor   shift1,x         ;Recombine the MSB with data
CC97:91 56       (6)   33         sta   (buffer2),y      ;Store it away
CC99:45 40       (3)   34         eor   checksum         ;Add it to the checksum
```

```
CC9B:85 40      (3)   35          sta   checksum
CC9D:C8         (2)   36          iny
CC9E:                 37 *
CC9E:                 38 * Now, the second Y turn over occurs at this point in the loop.
                        Update
CC9E:                 39 *  the buffer pointer if it occurred.
CC9E:                 40 *
CC9E:D0 02   CCA2(3)  41          bne   *+4
CCA0:E6 57      (5)   42          inc   buffer2+1
CCA2:                 43 *
CCA2:                 44 * Now the second byte
CCA2:                 45 *
CCA2:AD EC C0   (4)   46          lda   16clr+TheOff
CCA5:10 FB   CCA2(3)  47          bpl   *-3           ;Back 1 instruction
CCA7:5D AD CA   (4)   48          eor   shift2,x      ;Recombine the MSB with data
CCAA:91 56      (6)   49          sta   (buffer2),y   ;Store it away
CCAC:45 40      (3)   50          eor   checksum      ;Add it to the checksum
CCAE:85 40      (3)   51          sta   checksum
CCB0:C8         (2)   52          iny
CCB1:                 53 *
CCB1:                 54 * Now the third byte
CCB1:                 55 *
CCB1:AD EC C0   (4)   56          lda   16clr+TheOff
CCB4:10 FB   CCB1(3)  57          bpl   *-3           ;Back 1 instruction
CCB6:5D BD CA   (4)   58          eor   shift3,x      ;Recombine the MSB with data
CCB9:91 56      (6)   59          sta   (buffer2),y   ;Store it away
CCBB:45 40      (3)   60          eor   checksum      ;Add it to the checksum
CCBD:85 40      (3)   61          sta   checksum
CCBF:C8         (2)   62          iny
CCC0:                 63 *
CCC0:                 64 * Now the fourth byte
CCC0:                 65 *
CCC0:AD EC C0   (4)   66          lda   16clr+TheOff
CCC3:10 FB   CCC0(3)  67          bpl   *-3           ;Back 1 instruction
CCC5:5D CD CA   (4)   68          eor   shift4,x      ;Recombine the MSB with data
CCC8:91 56      (6)   69          sta   (buffer2),y   ;Store it away
CCCA:45 40      (3)   70          eor   checksum      ;Add it to the checksum
CCCC:85 40      (3)   71          sta   checksum
CCCE:C8         (2)   72          iny
CCCF:                 73 *
CCCF:                 74 * The first Y turn over occurs at this point in the loop.  Update
CCCF:                 75 *  the buffer pointer if it occurred.
CCCF:                 76 *
CCCF:D0 02   CCD3(3)  77          bne   *+4
CCD1:E6 57      (5)   78          inc   buffer2+1
CCD3:                 79 *
CCD3:A6 59      (3)   80          ldx   temp          ;Now we need the other index
CCD5:                 81 *
CCD5:                 82 * Now the fifth byte
CCD5:                 83 *
CCD5:AD EC C0   (4)   84          lda   16clr+TheOff
CCD8:10 FB   CCD5(3)  85          bpl   *-3           ;Back 1 instruction
CCDA:5D AD CA   (4)   86          eor   shift2,x      ;Recombine the MSB with data
CCDD:91 56      (6)   87          sta   (buffer2),y   ;Store it away
CCDF:45 40      (3)   88          eor   checksum      ;Add it to the checksum
CCE1:85 40      (3)   89          sta   checksum
CCE3:C8         (2)   90          iny
CCE4:                 91 *
CCE4:                 92 * Now the sixth byte
```

```
CCE4:                    93 *
CCE4:AD EC C0     (4)    94          lda   l6clr+TheOff
CCE7:10 FB   CCE4(3)     95          bpl   *-3              ;Back 1 instruction
CCE9:5D BD CA     (4)    96          eor   shift3,x         ;Recombine the MSB with data
CCEC:91 56        (6)    97          sta   (buffer2),y      ;Store it away
CCEE:45 40        (3)    98          eor   checksum         ;Add it to the checksum
CCF0:85 40        (3)    99          sta   checksum
CCF2:C8           (2)   100          iny
CCF3:                   101 *
CCF3:                   102 * And, finally, the seventh byte
CCF3:                   103 *
CCF3:AD EC C0     (4)   104          lda   l6clr+TheOff
CCF6:10 FB   CCF3(3)    105          bpl   *-3              ;Back 1 instruction
CCF8:5D CD CA     (4)   106          eor   shift4,x         ;Recombine the MSB with data
CCFB:91 56        (6)   107          sta   (buffer2),y      ;Store it away
CCFD:45 40        (3)   108          eor   checksum         ;Add it to the checksum
CCFF:85 40        (3)   109          sta   checksum
CD01:C8           (2)   110          iny
CD02:                   111 *
CD02:                   112 * Now see if this is the last group of seven to receive
CD02:                   113 *
CD02:C6 4B        (5)   114          dec   grp7ctr
CD04:F0 03   CD09(3)    115          beq   done5            ;Go to get the checksum etc
CD06:4C 7C CC     (3)   116          jmp   start35          ;Another topbits ...
CD09:                   117 *
CD09:                   118 * Get and reconstruct the checksum
CD09:                   119 *
CD09:         CD09      120 done5    equ   *
CD09:AD EC C0     (4)   121          lda   l6clr+TheOff
CD0C:10 FB   CD09(3)    122          bpl   *-3
CD0E:85 59        (3)   123          sta   temp             ;1 c6 1 c4 1 c2 1 c0
CD10:                   124 *
CD10:68           (4)   125          pla                    ;Restore groups of 7 counter
CD11:85 4B        (3)   126          sta   grp7ctr
CD13:AD EC C0     (4)   127          lda   l6clr+TheOff     ;1 c7 1 c5 1 c3 1 c1
CD16:10 FB   CD13(3)    128          bpl   *-3
CD18:38           (2)   129          sec
CD19:2A           (2)   130          rol   a                ;c7 1 c5 1 c3 1 c1 1
CD1A:25 59        (3)   131          and   temp             ;c7 c6 c5 c4 c3 c2 c1 c0
CD1C:45 40        (3)   132          eor   checksum         ;When we're done, should be zero
CD1E:                   133 *
CD1E:                   134 * Get the packet end mark. Is it correct?
CD1E:                   135 *
CD1E:AC EC C0     (4)   136 rdha5    ldy   l6clr+TheOff     ;Preserve A
CD21:10 FB   CD1E(3)    137          bpl   rdha5
CD23:                   138 *
CD23:C0 C8        (2)   139          cpy   #packetend
CD25:D0 1C   CD43(3)    140          bne   npenderr5
CD27:                   141 *
CD27:                   142 * Didn't have time before to checksum oddbytes.  Do it now
CD27:                   143 *  A still has the partial checksum
CD27:                   144 *
CD27:A6 4C        (3)   145          ldx   oddbytes
CD29:F0 08   CD33(3)    146          beq   icbt15
CD2B:A0 00        (2)   147          ldy   #0
CD2D:51 54        (5)   148 icbt5    eor   (buffer),y
CD2F:C8           (2)   149          iny
CD30:CA           (2)   150          dex
```

```
CD31:D0 FA   CD2D(3)  151         bne   icbt5
CD33:                 152 *
CD33:                 153 * Okay, checksum oughta be zero.  If not, checksum error.
CD33:                 154 *
CD33:        CD33     155 icbt15 equ   *
CD33:AA         (2)   156         tax
CD34:D0 11   CD47(3)  157         bne   cserror5
CD36:                 158 *
CD36:                 159 * Wait for /BSY to go low
CD36:                 160 *
CD36:        CD36     161 lstbsywait5 equ *
CD36:AD ED C0   (4)   162         lda   l6set+TheOff
CD39:AD EE C0   (4)   163 rdh45   lda   l7clr+TheOff
CD3C:30 FB   CD39(3)  164         bmi   rdh45
CD3E:                 165 *
CD3E:                 166 * Got the bytes, now acknowledge their reciept
CD3E:                 167 *
CD3E:AD E0 C0   (4)   168         lda   reqclr+TheOff ;Lower REQ
CD41:                 169 *
CD41:18         (2)   170         clc
CD42:60         (6)   171         rts
CD43:                 172 *
CD43:A9 20      (2)   173 npenderr5 lda #nopackend
CD45:D0 02   CD49(3)  174         bne   gserror5
CD47:A9 10      (2)   175 cserror5 lda #csumerr
CD49:38         (2)   176 gserror5 sec
CD4A:60         (6)   177         rts
CD4B:                 178 *
CD4B:                 139         fin
CD4B:                 140 *
CD4B:                 141         include pc.main
```

```
CD4B:                    2 *
CD4B:                    3 *
CD4B:          CD4B      4 Entry  equ   *
CD4B:90 03    CD50(3)    5        bcc   bentry        ;If non-boot, skip jump to boot
CD4D:4C 23 C5     (3)    6        jmp   bootcode
CD50:                    7 *
CD50:                    8 * X is still set to slot number.
CD50:                    9 *
CD50:          CD50     10 bentry equ   *
CD50:                   11 *
CD50:          0001     12        do    IIc*ROM
CD50:A9 40        (2)   13        lda   #%01000000
CD52:1C 78 04     (6)   14        trb   ProFlag+5     ;ProFlag is fixed in //c
CD55:                   15 *
CD55:          CD55     16 atentry equ  *
CD55:                   17        fin
CD55:                   18 *
CD55:D8           (2)   19        cld                 ;Don't want decimal mode!!
CD56:8A           (2)   20        txa
CD57:A8           (2)   21        tay                 ;Really want it in Y... no ROR ABS,Y!
CD58:                   22 *
CD58:                   23 * If this is a PC call, then get the address of the parm table
CD58:                   24 *
CD58:B9 73 04     (4)   25        lda   ProFlag,y
CD5B:30 11    CD6E(3)   26        bmi   noplay
CD5D:                   27 *
CD5D:68           (4)   28        pla                 ;Get lo order
CD5E:99 F3 05     (5)   29        sta   SHTempX,y     ;Keep lo parm address-1
CD61:18           (2)   30        clc
CD62:69 03        (2)   31        adc   #3
CD64:AA           (2)   32        tax                 ;Lo order new return address
CD65:68           (4)   33        pla                 ;Get hi order address
CD66:99 73 06     (5)   34        sta   SHTempY,y     ;Keep hi parm addr-1
CD69:69 00        (2)   35        adc   #0
CD6B:48           (3)   36        pha                 ;Push back new return address hi
CD6C:8A           (2)   37        txa
CD6D:48           (3)   38        pha                 ;Push new return address lo
CD6E:                   39 *
CD6E:          CD6E     40 noplay equ   *
CD6E:                   41 *
CD6E:                   42 * On the //c, it is important to have the Disk // enable lines
CD6E:                   43 *  off for as long as possible before using the IWM (phases, /WRREQ
CD6E:                   44 *  lines).  Wait here 'til the Disk // motors are off.
CD6E:                   45 *
CD6E:          0001     46        do    IIc
CD6E:20 35 CC     (6)   47        jsr   WaitIWMOff    ;Must preserve Y!!
CD71:                   48        fin
CD71:                   49 *
CD71:                   50 * We can't really tolerate interrupts in most of the code, so
                           disable
CD71:                   51 *
CD71:08           (3)   52        php                 ;Save interrupt status
CD72:78           (2)   53        sei                 ;No interrupts please
CD73:                   54 *
CD73:                   55 * Preserve the zero page work area
CD73:                   56 *
CD73:A2 1B        (2)   57        ldx   #ZPSize-1
CD75:B5 40        (4)   58 pzp    lda   ZeroPage,x
CD77:48           (3)   59        pha
```

```
CD78:CA          (2)   60          dex
CD79:10 FA   CD75(3)   61          bpl   pzp
CD7B:                  62 *
CD7B:                  63 * Okay, we're safe... now it's all right to store in zero page
CD7B:                  64 *
CD7B:84 58       (3)   65          sty   Slot
CD7D:                  66 *
CD7D:        0001      67          ifeq  IIc^ROM
CD7D:                  80          fin
CD7D:                  81 *
CD7D:                  82 * Now map any ProDOS unit references to our sequential ones.
CD7D:                  83 *  The method is bizzare and magicians never reveal their secrets.
CD7D:                  84 *
CD7D:        CD7D      85 allset equ   *
CD7D:A5 43       (3)   86          lda   CMDUnit       ;76543210 7&6 specify unit
CD7F:2A          (2)   87          rol   a             ;6543210X C<-7
CD80:08          (3)   88          php                 ;Save drive num
CD81:2A          (2)   89          rol   a             ;543210X7 C<-6
CD82:2A          (2)   90          rol   a             ;43210X76 (6 is grp of 2)
CD83:28          (4)   91          plp                 ;C<-7
CD84:2A          (2)   92          rol   a             ;3210X767
CD85:29 03       (2)   93          and   #%00000011    ;ProDOS only installs up to 4
CD87:49 02       (2)   94          eor   #%00000010    ;000000/67; 6 was /grpoftwo
CD89:C0 04       (2)   95          cpy   #4            ;If in slot 1,2,or3 reverse grps of two
CD8B:B0 02   CD8F(3)   96          bge   allset1
CD8D:49 02       (2)   97          eor   #%00000010
CD8F:AA          (2)   98 allset1 tax
CD90:E8          (2)   99          inx
CD91:86 43       (3)  100          stx   CMDUnit       ;You got it
CD93:                 101 *
CD93:                 102 * Now if this is through the MLI xface, gotta copy stuff into the
CD93:                 103 *  send buffer from the parameter list.
CD93:                 104 *
CD93:B9 73 04    (4)  105          lda   ProFlag,y
CD96:10 03   CD9B(3)  106          bpl   darnit
CD98:4C 3F CE    (3)  107          jmp   skipcopy
CD9B:                 108 *
CD9B:                 109 * Get the address of the in-line parameter table
CD9B:                 110 *
CD9B:        CD9B     111 darnit equ   *
CD9B:B9 F3 05    (4)  112          lda   SHTempX,y     ;Get back the low part buff addr
CD9E:85 54       (3)  113          sta   buffer
CDA0:B9 73 06    (4)  114          lda   SHTempY,y     ; and the hi part
CDA3:85 55       (3)  115          sta   buffer+1
CDA5:                 116 *
CDA5:                 117 * Now pull out the command code, and the address of the parameters.
CDA5:                 118 *
CDA5:A0 01       (2)  119          ldy   #1            ;Stacked address is EA-1
CDA7:B1 54       (5)  120          lda   (buffer),y
CDA9:85 42       (3)  121          sta   cmdcode       ;Nice
CDAB:C8          (2)  122          iny
CDAC:B1 54       (5)  123          lda   (buffer),y    ;Get lo part of parmlist address
CDAE:AA          (2)  124          tax                 ;Save it
CDAF:C8          (2)  125          iny
CDB0:B1 54       (5)  126          lda   (buffer),y    ;Get hi part
CDB2:85 55       (3)  127          sta   buffer+1
CDB4:86 54       (3)  128          stx   buffer
CDB6:                 129 *
```

```
CDB6:                    130 * Now buffer points to parmlist
CDB6:                    131 *  Check command type, and pidgeonhole the parmlist length
CDB6:                    132 *
CDB6:A9 01        (2)    133          lda   #BadCmd
CDB8:A6 42        (3)    134          ldx   cmdcode
CDBA:E0 0A        (2)    135          cpx   #$A              ;Only valid codes are 0-9
CDBC:90 03    CDC1(3)    136          blt   noeh             ;=> at least he got that right
CDBE:4C 0F CF     (3)    137 Errorhitch jmp Error           ;Gee, maybe we should promote this guy...
CDC1:         CDC1       138 noeh     equ   *
CDC1:A0 00        (2)    139          ldy   #0               ;Set for indct compare
CDC3:B1 54        (5)    140          lda   (buffer),y       ;Get # of parms?
CDC5:85 5A        (3)    141          sta   Unit
CDC7:                    142 *
CDC7:                    143 * Now copy the bytes
CDC7:                    144 *
CDC7:         CDC7       145 okaycnt equ  *
CDC7:A0 08        (2)    146          ldy   #>cmdlength-1 ;Always copy the maximum
CDC9:         CDC9       147 copyloop equ *
CDC9:B1 54        (5)    148          lda   (buffer),y       ;Pull it out of their hat
CDCB:99 42 00     (5)    149          sta   cmdcode,y        ;Stuff it into mine
CDCE:88           (2)    150          dey
CDCF:D0 F8    CDC9(3)    151          bne   copyloop         ;Copy 'em all
CDD1:                    152 *
CDD1:                    153 * Okay.  The caller of the PC could be making one of three calls
CDD1:                    154 *  with a unit number of $00, Control, Init or Status.  Check for
CDD1:                    155 *  these and do what is appropriate.
CDD1:                    156 *
CDD1:A5 43        (3)    157          lda   CMDUnit
CDD3:D0 6A    CE3F(3)    158          bne   skipcopy         ;Never mind
CDD5:                    159 *
CDD5:                    160 * Check the parameter count for this call to unit#0
CDD5:                    161 *
CDD5:A6 42        (3)    162          ldx   CMDCode
CDD7:BD 86 CF     (4)    163          lda   parmctab,x       ;Get the length this command
CDDA:29 7F        (2)    164          and   #$7F             ;Force 0 -> MSB
CDDC:A8           (2)    165          tay                    ;Hang on
CDDD:A9 04        (2)    166          lda   #BadPCnt         ;Antic bad count
CDDF:C4 5A        (3)    167          cpy   Unit             ;User's pcount is currently here
CDE1:D0 DB    CDBE(3)    168          bne   ErrorHitch       ;What a baby!
CDE3:                    169 *
CDE3:                    170 * Now service one of the three commands
CDE3:                    171 *
CDE3:E0 05        (2)    172          cpx   #InitCMD
CDE5:D0 0A    CDF1(3)    173          bne   notinit          ;Not an Init call
CDE7:A9 00        (2)    174          lda   #PowerReset      ;Just like powerup or reset key(//c)
CDE9:20 90 CF     (6)    175          jsr   AssignID         ;Do a reset cycle
CDEC:A9 00        (2)    176 Aokay    lda   #0               ;No error allowed
CDEE:4C 31 CF     (3)    177          jmp   sa2
CDF1:                    178 *
CDF1:8A           (2)    179 notinit  txa                    ;Equiv to 'cmp #StatusCMD'
CDF2:D0 24    CE18(3)    180          bne   maybectrl
CDF4:                    181 *
CDF4:A9 21        (2)    182          lda   #BadCtl          ;Antic a non zero stat code
CDF6:A6 46        (3)    183          ldx   CMDSCode         ;Stat unit#0 can only be code=0
CDF8:D0 C4    CDBE(3)    184          bne   ErrorHitch
CDFA:                    185 *
CDFA:8A           (2)    186          txa                    ;Equiv to 'lda #0'
CDFB:A6 58        (3)    187          ldx   Slot
```

```
CDFD:A0 07      (2)  188        ldy   #7
CDFF:91 44      (6)  189 nin1   sta   (CmdBuffer1),y ;Clear some space
CE01:88         (2)  190        dey
CE02:D0 FB  CDFF(3)  191        bne   nin1
CE04:                192 *
CE04:BD F9 06   (4)  193        lda   NumDevices,x
CE07:91 44      (6)  194        sta   (CMDBuffer1),y ;Stick it where they want it
CE09:C8         (2)  195        iny
CE0A:                196 *
CE0A:       0001     197        do    IIc
CE0A:AD F9 04   (4)  198        lda   $4F9          ;//c Port 1 interrupt status
CE0D:                199        else
CE0D:                201        fin
CE0D:                202 *
CE0D:91 44      (6)  203        sta   (CMDBuffer1),y ;Store PC interrupt status
CE0F:                204 *
CE0F:A9 08      (2)  205        lda   #8
CE11:88         (2)  206        dey                 ;A,Y has 0008; # bytes status
CE12:20 F2 CF   (6)  207        jsr   squirrel
CE15:                208 *
CE15:4C EC CD   (3)  209        jmp   Aokay         ;Skip down (up) with no error
CE18:       CE18     210 maybectrl equ *
CE18:C9 04      (2)  211        cmp   #ControlCMD
CE1A:D0 0B  CE27(3)  212        bne   BUnit         ;Unit #0 was a bad one
CE1C:                213 *
CE1C:A6 46      (3)  214        ldx   CMDSCode      ;We allow two control calls for Unit#0
CE1E:F0 0B  CE2B(3)  215        beq   enabint       ;0 means enable interrupts
CE20:CA         (2)  216        dex
CE21:F0 14  CE37(3)  217        beq   disabint      ;1 means disable interrupts
CE23:A9 21      (2)  218        lda   #badctl
CE25:       CE25     219 ErrorHitch2 equ *
CE25:D0 97  CDBE(3)  220        bne   ErrorHitch    ;No other codes allowed
CE27:                221 *
CE27:       CE27     222 BUnit  equ   *
CE27:A9 11      (2)  223        lda   #badUnit      ;Only certain calls can have Unit#0
CE29:D0 93  CDBE(3)  224        bne   ErrorHitch    ;Branch always
CE2B:                225 *
CE2B:       0001     226        do    IIc
CE2B:       CE2B     227 enabint equ  *
CE2B:A9 C0      (2)  228        lda   #$C0
CE2D:8D F9 05   (4)  229        sta   $5F9
CE30:A9 0F      (2)  230        lda   #$0F
CE32:0C 9A C0   (6)  231        tsb   $C09A
CE35:D0 05  CE3C(3)  232        bne   aokayhitch
CE37:                233 *
CE37:       CE37     234 disabint equ *
CE37:A9 01      (2)  235        lda   #$01
CE39:1C 9A C0   (6)  236        trb   $C09A
CE3C:4C EC CD   (3)  237 aokayhitch jmp AOkay
CE3F:                238 *
CE3F:                239        else
CE3F:                244        fin
CE3F:                245 *
CE3F:                246 * Okay, everything's all groovy.  ProDOS re-enters here.
CE3F:                247 *  Check Unit number to be sure there is a corresponding device
CE3F:                248 *
CE3F:       CE3F     249 skipcopy equ *
CE3F:A9 28      (2)  250        lda   #NoDrive      ;Anticpate bad unit number
```

```
CE41:A4 58       (3)  251         ldy   slot
CE43:BE F9 06    (4)  252         ldx   NumDevices,y
CE46:E4 43       (3)  253         cpx   CMDUnit
CE48:90 DB   CE25(3)  254         blt   ErrorHitch2   ;Safe- If C clr then Z is clr
CE4A:                 255 *
CE4A:                 256 * Set buffer and bytecount in anticpation of the inevitable
CE4A:                 257 * SendPack.
CE4A:A9 09       (2)  258         lda   #>cmdlength
CE4C:85 4D       (3)  259         sta   bytecount1
CE4E:A9 00       (2)  260         lda   #<cmdlength
CE50:85 4E       (3)  261         sta   bytecounth
CE52:85 55       (3)  262         sta   buffer+1
CE54:A9 42       (2)  263         lda   #>cmdcode
CE56:85 54       (3)  264         sta   buffer
CE58:                 265 *
CE58:                 266 * If it's a PC call, omit the next two steps
CE58:                 267 *
CE58:A6 58       (3)  268         ldx   Slot
CE5A:BD 73 04    (4)  269         lda   ProFlag,x     ;Is it a call from ProDOS?
CE5D:10 13   CE72(3)  270         bpl   notstat       ;=> Statcode already set...
CE5F:                 271 *
CE5F:                 272 * Need to generate a parameter count for a ProDOS call
CE5F:                 273 *
CE5F:A6 42       (3)  274         ldx   CMDCode
CE61:BD 86 CF    (4)  275         lda   ParmCTab,x
CE64:29 7F       (2)  276         and   #$7F
CE66:85 5A       (3)  277         sta   Unit
CE68:                 278 *
CE68:                 279 * ProDOS always needs the highest blockno byte zeroed
CE68:                 280 *
CE68:A9 00       (2)  281         lda   #0
CE6A:85 48       (3)  282         sta   CMDBlockS
CE6C:                 283 *
CE6C:                 284 * If this is a ProDOS status call, set stat code to zero
CE6C:                 285 *
CE6C:A5 42       (3)  286         lda   CMDCode
CE6E:D0 02   CE72(3)  287         bne   notstat       ;=> Not status so forget it
CE70:                 288 *lda #SCDeviceStat ;A is already zero
CE70:85 46       (3)  289         sta   CMDSCode      ;Store in command table
CE72:                 290 *
CE72:                 291 * Okay, finally send over the damn command
CE72:                 292 *
CE72:        CE72     293 notstat equ  *
CE72:A5 5A       (3)  294         lda   Unit
CE74:A6 43       (3)  295         ldx   CmdPCount     ;Swap the Parmcount & unit#
CE76:86 5A       (3)  296         stx   Unit
CE78:85 43       (3)  297         sta   CMDPCount     ;Now they're correct
CE7A:                 298 *
CE7A:A9 80       (2)  299         lda   #cmdmark
CE7C:85 5B       (3)  300         sta   WPacketType
CE7E:                 301 *
CE7E:20 8A CA    (6)  302         jsr   ClrPhases     ;Bring all phases off
CE81:                 303 *
CE81:20 EC CA    (6)  304         jsr   SendPack
CE84:B0 46   CECC(3)  305         bcs   behitch       ;If not okay, skip to bus error
CE86:                 306 *
CE86:                 307 * Now copy over the buffer address for any data xfer.
CE86:                 308 *
```

```
CE86:A5 44      (3)  309        lda   CMDBuffer
CE88:85 54      (3)  310        sta   buffer
CE8A:A5 45      (3)  311        lda   CMDBuffer+1
CE8C:85 55      (3)  312        sta   buffer+1
CE8E:                313 *
CE8E:                314 * Now for some commands, we have to send over a packet of data, too.
CE8E:                315 *  See if this command is one of THOSE.
CE8E:                316 *
CE8E:A6 42      (3)  317        ldx   cmdcode
CE90:BD 86 CF   (4)  318        lda   parmctab,x
CE93:10 3B   CED0(3) 319        bpl   noxtrasend    ;Encoded in top bit
CE95:                320 *
CE95:                321 * The buffer address and bytecount depend on the call type.
CE95:                322 *
CE95:E0 04      (2)  323        cpx   #ControlCmd
CE97:D0 18   CEB1(3) 324        bne   NOControl
CE99:                325 *
CE99:                326 * In the case of control, bytecount:=(buffer) then buffer:=buffer+2
CE99:                327 *
CE99:A0 01      (2)  328        ldy   #1
CE9B:B1 54      (5)  329        lda   (buffer),y    ;Get Hi order bytecount
CE9D:AA         (2)  330        tax
CE9E:88         (2)  331        dey
CE9F:B1 54      (5)  332        lda   (buffer),y
CEA1:48         (3)  333        pha                 ;Keep for later
CEA2:18         (2)  334        clc
CEA3:A9 02      (2)  335        lda   #2
CEA5:65 54      (3)  336        adc   buffer
CEA7:85 54      (3)  337        sta   buffer
CEA9:68         (4)  338        pla                 ;Get back Lo order bytecount
CEAA:90 13   CEBF(3) 339        bcc   secondsend    ;Skip hi ord increment
CEAC:E6 55      (5)  340        inc   buffer+1
CEAE:4C BF CE   (3)  341        jmp   secondsend    ;Skip to store bytecount
CEB1:                342 *
CEB1:        CEB1    343 NOControl equ *
CEB1:E0 02      (2)  344        cpx   #WriteCMD     ;Check for a writeblock
CEB3:D0 06   CEBB(3) 345        bne   NOWBlock      ;Must be control or write
CEB5:                346 *
CEB5:                347 * In the case of WriteBlock, the length is 512 and the buffer
CEB5:                348 *  address is at buffer in the command table
CEB5:                349 *
CEB5:A9 00      (2)  350        lda   #0
CEB7:A2 02      (2)  351        ldx   #2
CEB9:D0 04   CEBF(3) 352        bne   secondsend
CEBB:                353 *
CEBB:                354 * For FileWrite, the buffer address is at CMDbuffer
CEBB:                355 *  and the length is at CMDblock.
CEBB:                356 *
CEBB:        CEBB    357 NOWBlock equ *
CEBB:A6 47      (3)  358        ldx   CMDBlockh
CEBD:A5 46      (3)  359        lda   CMDBlockl
CEBF:                360 *
CEBF:        CEBF    361 secondsend equ *
CEBF:86 4E      (3)  362        stx   bytecounth
CEC1:85 4D      (3)  363        sta   bytecountl
CEC3:                364 *
CEC3:A9 82      (2)  365        lda   #datamark
CEC5:85 5B      (3)  366        sta   WPacketType   ;Identify this as a data packet
```

```
CEC7:                   367 *
CEC7:20 DD CA    (6)    368         jsr   SendData
CECA:90 04   CED0(3)    369         bcc   noxtrasend
CECC:        CECC       370 behitch equ   *
CECC:A9 06       (2)    371         lda   #BusErr       ;This is the bus error hitch
CECE:D0 3F   CF0F(3)    372         bne   Error
CED0:                   373 *
CED0:                   374 * On ProDOS status call, we've got to point the buffer pointer
CED0:                   375 *  correctly to zero page... it's the only case special case
CED0:                   376 *  (on Write, Format and Control no data comes back).
CED0:                   377 *
CED0:        CED0       378 noxtrasend equ *
CED0:A4 58       (3)    379         ldy   Slot
CED2:B9 73 04    (4)    380         lda   ProFlag,y
CED5:10 0C   CEE3(3)    381         bpl   getresults
CED7:A5 42       (3)    382         lda   cmdcode
CED9:D0 08   CEE3(3)    383         bne   getresults
CEDB:                   384 *
CEDB:A9 45       (2)    385         lda   #>CMDBufferh  ;Want status in these four
CEDD:A2 00       (2)    386         ldx   #<CMDBufferh
CEDF:85 54       (3)    387         sta   buffer
CEE1:86 55       (3)    388         stx   buffer+1
CEE3:                   389 *
CEE3:                   390 * Please to be calling ReceivePack
CEE3:                   391 *
CEE3:        CEE3       392 getresults equ *
CEE3:20 33 CB    (6)    393         jsr   RecPack       ;Get status byte (maybe read data too)
CEE6:B0 E4   CECC(3)    394         bcs   behitch
CEE8:                   395 *
CEE8:                   396 * Figure how many bytes were sent and put that in X,Y temps
CEE8:                   397 *
CEE8:20 50 CC    (6)    398         jsr   Rcvcount      ;Do the times 7...
CEEB:20 F2 CF    (6)    399         jsr   squirrel      ;Store away count in SHTEMPs
CEEE:                   400 *
CEEE:                   401 * For the ProDOS status call, we've got to look at the status byte
CEEE:                   402 *  returned and return a DIP error if appropriate.
CEEE:                   403 *  Also overwrite the X,Y temps with # blocks if this is a ProDOS
CEEE:                   404 * Stat call.
CEEE:A5 42       (3)    405         lda   CMDCode       ;Is it a ProDOS status call
CEF0:D0 1B   CF0D(3)    406         bne   noerror
CEF2:A6 58       (3)    407         ldx   Slot
CEF4:BD 73 04    (4)    408         lda   ProFlag,x
CEF7:10 14   CF0D(3)    409         bpl   noerror
CEF9:                   410 *
CEF9:A5 46       (3)    411         lda   CMDBlockl     ;This'll get loaded into the XY regs
                                                          later
CEFB:9D F3 05    (5)    412         sta   SHTempX,x
CEFE:A5 47       (3)    413         lda   CMDBlockh
CF00:9D 73 06    (5)    414         sta   SHTempY,x
CF03:                   415 *
CF03:A5 45       (3)    416         lda   CMDBufferh    ;Check status byte
CF05:29 10       (2)    417         and   #SVMask1
CF07:D0 04   CF0D(3)    418         bne   noerror       ;No DIP
CF09:A9 2F       (2)    419         lda   #OffLine
CF0B:D0 02   CF0F(3)    420         bne   Error
CF0D:                   421 *
CF0D:                   422 * Now it's time to think about returning to the caller
CF0D:                   423 *  Remember that ProDOS doesn't want to know about soft errors,
CF0D:                   424 *  only fatal ones.  If this is a ProDOS call, and the soft error
```

```
CF0D:                   425 *  bit in the statbyte is set, there IS NO error (statbyte is
                               cleared).
CF0D:                   426 *  Also, ProDOS wants only I/O, Write Protect, No Device, Offline.
CF0D:                   427 *  If any other hard error comes from the device on a ProDOS call,
CF0D:                   428 *  map it to an I/O Error.  (Gross me out.)
CF0D:                   429 *
CF0D:          CF0D     430 noerror equ  *
CF0D:A5 4D       (3)    431         lda   statbyte
CF0F:          CF0F     432 Error   equ   *
CF0F:A4 58       (3)    433         ldy   Slot            ;Need access to screenholes
CF11:99 F3 04    (5)    434         sta   Retry,Y         ;Keep unadulterated error in shole
CF14:AA          (2)    435         tax                   ;Set the Z flag
CF15:F0 1A   CF31(3)    436         beq   sa2             ;Special case the zero
CF17:                   437 *
CF17:BE 73 04    (4)    438         ldx   ProFlag,y       ;Set N to ProDOS call or not
CF1A:10 15   CF31(3)    439         bpl   sa2             ;If PC call, no mapping occurs
CF1C:                   440 *
CF1C:A2 00       (2)    441         ldx   #0              ;Assume a soft error
CF1E:C9 40       (2)    442         cmp   #%01000000      ;Soft error check
CF20:B0 0E   CF30(3)    443         bge   storeaway       ;If $40 or bigger, map to zero
CF22:                   444 *
CF22:A2 27       (2)    445         ldx   #IOError        ;Now anticipate ProDOS I/O error
CF24:C9 2B       (2)    446         cmp   #WriteProt
CF26:F0 09   CF31(3)    447         beq   sa2             ;OK to return Write Protect
CF28:C9 28       (2)    448         cmp   #NoDrive
CF2A:F0 05   CF31(3)    449         beq   sa2             ;OK to return Drive disconnected
CF2C:C9 2F       (2)    450         cmp   #OffLine
CF2E:F0 01   CF31(3)    451         beq   SA2
CF30:                   452 *
CF30:          CF30     453 storeaway equ *
CF30:8A          (2)    454         txa                   ;Use the default value
CF31:          CF31     455 sa2     equ   *
CF31:A4 58       (3)    456         ldy   Slot
CF33:99 73 05    (5)    457         sta   SHTemp1,y       ;Keep in screenhole
CF36:                   458 *
CF36:                   459 * If this is the //c version, we need to reset the IWM to its
CF36:                   460 *  former disk // state.  This is done by setting the mode register
CF36:                   461 *  to a little known (and less documented) mode which speeds up the
CF36:                   462 *  internal motor timeout.  When the motor enable has timed out,
CF36:                   463 *  mode can be set back to zero.  This method is necessary because
CF36:                   464 *  if the timer is enabled within the timeout period, the motor on
CF36:                   465 *  a Rev A IWM pops on for the full timeout period (since mode
                               changes
CF36:                   466 *  are disabled when the motor is on.  I know, it's bizzarre. Blame
                               Mac.
CF36:          0001     467          do    IIc
CF36:AD E8 C0    (4)    468          lda   monclr+$60      ;Motor off
CF39:2C ED C0    (4)    469          bit   16set+$60       ;Into mode reg access mode
CF3C:A9 2B       (2)    470          lda   #$2B            ;This is the magic "speed up" value
CF3E:8D EF C0    (4)    471          sta   17set+$60       ;Throw into mode register
CF41:EA          (2)    472          nop                   ;You're supposed to wait a while
CF42:EA          (2)    473          nop
CF43:EA          (2)    474          nop
CF44:EA          (2)    475          nop
CF45:          CF45     476 waitoff equ  *
CF45:AD EE C0    (4)    477          lda   17clr+$60       ;Wait 'til motor off
CF48:29 20       (2)    478          and   #$20
CF4A:D0 F9   CF45(3)    479          bne   waitoff
CF4C:A0 00       (2)    480          ldy   #0              ;Now set the reg back to $00
CF4E:A2 60       (2)    481          ldx   #$60            ;IWM's in slot 6
CF50:20 1F CC    (6)    482          jsr   SetIWMode
```

```
CF53:AD EC C0    (4)  483         lda   16clr+$60
CF56:AD E2 C0    (4)  484         lda   ca1clr+$60
CF59:AD E6 C0    (4)  485         lda   1strbclr+$60
CF5C:A4 58       (3)  486         ldy   Slot            ;Need Slot in Y
CF5E:                 487         fin
CF5E:                 488 *
CF5E:                 489 * Now, restore our zero page area.
CF5E:                 490 *
CF5E:A2 00       (2)  491         ldx   #0
CF60:68          (4)  492 rzp     pla
CF61:95 40       (4)  493         sta   zeropage,x
CF63:E8          (2)  494         inx
CF64:E0 1C       (2)  495         cpx   #ZPSize
CF66:90 F8   CF60(3)  496         blt   rzp
CF68:                 497 *
CF68:                 498 * We're into the stretch!  Restore interrupt mask, load X, Y, and A
CF68:                 499 *  and set the carry if the error byte is non-zero.
CF68:                 500 *
CF68:28          (4)  501         plp                   ;Restore interrupt flag
CF69:B9 F3 05    (4)  502         lda   SHTempx,y       ;Get X value
CF6C:AA          (2)  503         tax
CF6D:B9 73 05    (4)  504         lda   SHTemp1,y       ;Grab the error result code
CF70:48          (3)  505         pha
CF71:B9 73 06    (4)  506         lda   SHTempy,y       ;Pull out the Y value
CF74:A8          (2)  507         tay                   ;No more access to screenholes
CF75:18          (2)  508         clc                   ;Anticipate zero result code
CF76:68          (4)  509         pla                   ;Pull back result code
CF77:F0 01   CF7A(3)  510         beq   finalskip       ;Return with carry clear
CF79:38          (2)  511         sec                   ;Some type of error
CF7A:         CF7A    512 finalskip equ *
CF7A:                 513 *
CF7A:         0001    514         do    IIc^ROM
CF7A:08          (3)  515         php                   ;Save carry and Z flag
CF7B:2C 78 04    (4)  516         bit   ProFlag+5       ;Ick - ProFlag is fixed in //c
CF7E:70 04   CF84(3)  517         bvs   ick1            ;If bit 6=1, then return to alt ROM
CF80:28          (4)  518         plp                   ;Vclr so return across ROM bank bdy
CF81:4C 84 C7    (3)  519         jmp   SWRTS2
CF84:         CF84    520 ick1    equ   *
CF84:28          (4)  521         plp
CF85:60          (6)  522         rts                   ;Flags set correctly again
CF86:                 523         else
CF86:                 525         fin
CF86:                 526 *
CF86:                 527 *
CF86:         CF86    528 parmctab equ *
CF86:03               529         dfb   %00000011       ;Status:  3 parms/no data send
CF87:03               530         dfb   %00000011       ;Read:    3 parms/no data send
CF88:83               531         dfb   %10000011       ;Write:   3 parms/data send
CF89:01               532         dfb   %00000001       ;Format:  1 parm /no data send
CF8A:83               533         dfb   %10000011       ;Control: 3 parms/data send
CF8B:01               534         dfb   %00000001       ;Init:    1 parm /no data send
CF8C:01               535         dfb   %00000001       ;Open:    1 parm /no data send
CF8D:01               536         dfb   %00000001       ;Close:   1 parm /no data send
CF8E:03               537         dfb   %00000011       ;CharRead: 3 parms/data send
CF8F:83               538         dfb   %10000011       ;CharWrite: 3 parms/data send
CF90:                 539 *
CF90:                 540 *
```

```
CF90:                   542 *
CF90:          CF90     543 AssignID equ *
CF90:48           (3)   544         pha                 ;Save the init code
CF91:20 60 CA     (6)   545         jsr   resetchain    ;Reset all of those things
CF94:68           (4)   546         pla
CF95:AA           (2)   547         tax                 ;Save InitCode
CF96:                   548 *
CF96:                   549 * Save the command code, unit, and init code 'cause we'll trample
CF96:                   550 * 'em.
CF96:A5 42        (3)   551         lda   CMDCode
CF98:48           (3)   552         pha
CF99:A5 43        (3)   553         lda   CMDPCount
CF9B:48           (3)   554         pha
CF9C:A5 46        (3)   555         lda   CMDSCode
CF9E:48           (3)   556         pha
CF9F:86 46        (3)   557         stx   CMDSCode      ;Store away the type of INIT
CFA1:                   558 *
CFA1:                   559 * Set up to send DefID command packets
CFA1:                   560 *
CFA1:A9 05        (2)   561         lda   #InitCmd
CFA3:85 42        (3)   562         sta   CMDCode
CFA5:A9 00        (2)   563         lda   #0
CFA7:85 5A        (3)   564         sta   Unit
CFA9:A9 02        (2)   565         lda   #2            ;# parms in Init call
CFAB:85 43        (3)   566         sta   CMDPCount
CFAD:                   567 *
CFAD:                   568 * Point the buffer pointer
CFAD:                   569 *
CFAD:A9 42        (2)   570         lda   #>CMDCode
CFAF:85 54        (3)   571         sta   buffer
CFB1:A9 00        (2)   572         lda   #<CMDCode
CFB3:85 55        (3)   573         sta   buffer+1
CFB5:A9 80        (2)   574         lda   #cmdmark
CFB7:85 5B        (3)   575         sta   WPacketType
CFB9:                   576 *
CFB9:20 8A CA     (6)   577         jsr   ClrPhases     ;Make sure phases are off
CFBC:                   578 *
CFBC:                   579 * Send an ID for the next device in the chain
CFBC:                   580 *
CFBC:          CFBC     581 mordevices equ *
CFBC:E6 5A        (5)   582         inc   Unit
CFBE:A9 09        (2)   583         lda   #>cmdlength
CFC0:85 4D        (3)   584         sta   bytecountl    ;ReceivePack scrambles count
CFC2:A9 00        (2)   585         lda   #<cmdlength
CFC4:85 4E        (3)   586         sta   bytecounth
CFC6:                   587 *
CFC6:20 86 C8     (6)   588         jsr   SendOnePack   ;Send the command
CFC9:90 05    CFD0(3)   589         bcc   mdev2         ;If okay, skip to get response
CFCB:                   590 *
CFCB:C6 5A        (5)   591         dec   Unit
CFCD:4C D7 CF     (3)   592         jmp   mdev1
CFD0:                   593 *
CFD0:20 E6 C9     (6)   594 mdev2   jsr   ReceivePack   ;Get the response
CFD3:A5 4D        (3)   595         lda   statbyte
CFD5:F0 E5    CFBC(3)   596         beq   mordevices
CFD7:                   597 *
CFD7:                   598 * Okay, we done last device.  Squirrel away the number of devices.
CFD7:                   599 *
```

```
CFD7:A5 5A      (3)  600 mdev1  lda   Unit
CFD9:A4 58      (3)  601        ldy   slot
CFDB:99 F9 06   (5)  602        sta   NumDevices,y  ;Devices out there
CFDE:                603 *
CFDE:                604 * Recover the scrambled ProDOS parms
CFDE:                605 *
CFDE:68         (4)  606        pla
CFDF:85 46      (3)  607        sta   CMDSCode
CFE1:68         (4)  608        pla
CFE2:85 43      (3)  609        sta   CMDPCount
CFE4:68         (4)  610        pla
CFE5:85 42      (3)  611        sta   CMDCode
CFE7:                612 *
CFE7:       0001     613        ifeq  IIc^ROM
CFE7:                622        fin
CFE7:60         (6)  623        rts
CFE8:                624 *
CFE8:                625 *
CFE8:       0001     626        do    IIc
CFE8:       CFE8     627 AppleTalkEntry equ *
CFE8:                628 *
CFE8:                629 * This is an entry for the //c AppleTalk stump.
CFE8:                630 *
CFE8:A2 05      (2)  631        ldx   #5
CFEA:A9 40      (2)  632        lda   #%01000000    ;PC call & return to alt ROM
CFEC:9D 73 04   (5)  633        sta   ProFlag,x
CFEF:4C 55 CD   (3)  634        jmp   atentry       ;Just like normal
CFF2:                635        fin
CFF2:                636 *
CFF2:                637 *
CFF2:       CFF2     638 squirrel equ *
CFF2:A6 58      (3)  639        ldx   Slot
CFF4:9D F3 05   (5)  640        sta   SHTempX,x
CFF7:98         (2)  641        tya
CFF8:9D 73 06   (5)  642        sta   SHTempY,x
CFFB:60         (6)  643        rts
CFFC:                644 *
CFFC:                645 *
CFFC:                142 *
CFFC:       0001     143        ifeq  IIc^ROM
CFFC:                145        fin
CFFC:                146 *
CFFC:       CFFC     147 zzzzz  equ   *
CFFC:       0001     148        ifeq  IIc^ROM       ;If not //c ROM, pad bytes
CFFC:                153        fin
CFFC:                154 *
```

```
 C90E ACHE1          CD8F ALLSET1        ?CD7D ALLSET        ?CB04 ALTSENDPILE
 CE3C AOKAYHITCH     CDEC AOKAY           CFE8 APPLETALKENTRY CF90 ASSIGNID
 CD55 ATENTRY       ?FABA AUTOSCAN        CB61 AUXPTRINC        56 AUXPTR
?  4E AUXTYPE       ?  2D BADBLOCK          01 BADCMD        ?  22 BADCTLPARM
   21 BADCTL           04 BADPCNT           11 BADUNIT       ?E000 BASIC
 C52F BC1            CECC BEHITCH         CD50 BENTRY         CC28 BIZ
 0011 BMSGLEN        C521 BOOTC          ?C514 BOOTCASE5      C523 BOOTCODE
 C552 BOOTFAIL       C55F BOOTMSG         07DB BOOTSCRN       C570 BOOTTAB
   32 BSYTO1           0A BSYTO2            54 BUFFER           56 BUFFER2
 CE27 BUNIT            06 BUSERR         ?  40 BUSHOG        ?  08 BYTECMP
   4D BYTECOUNT        4E BYTECOUNTH        4D BYTECOUNTL    ?C500 C500ORG
 C082 CA1CLR         C083 CA1SET          C084 CA2CLR         C085 CA2SET
 CC2C CAREFUL        C8A5 CHAINUNBSY        40 CHECKSUM    ?  24 CH
 CFFF CLEARIOROMS    CA8A CLRPHASES         47 CMDBLOCKH        46 CMDBLOCKL
   48 CMDBLOCKS   ?  46 CMDBLOCK           45 CMDBUFFERH       44 CMDBUFFERL
   44 CMDBUFFER        42 CMDCODE           09 CMDLENGTH     ?C58A CMDLIST
   80 CMDMARK          43 CMDPCOUNT         46 CMDSCODE      ?  49 CMDSPARE1
?  4A CMDSPARE2        43 CMDUNIT         C55D COMA             80 COMMRESET
   04 CONTROLCMD     CDC9 COPYLOOP       ?FDED COUT          CD47 CSERROR5
   10 CSUMERR        ?  25 CV             CD9B DARNIT           82 DATAMARK
 C999 DATDONE         C9C3 DBERROR       ?CBE1 DETTOPBITS    ?  50 DEVICEID
 CE37 DISABINT        CB55 DIV7TAB         CB9E DIVIDE1        CBA7 DIVIDE2
 CB8E DIVIDE3         CB95 DIVIDE4         CBB0 DIVIDE5     ?CB64 DIVIDE7
 CD09 DONE5           CE2B ENABINT       ?C08A ENABLE1         C08B ENABLE2
 CA80 ENABLECHAIN     CD4B ENTRY           CF0F ERROR          CDBE ERRORHITCH
 CE25 ERRORHITCH2     CF7A FINALSKIP    ?  03 FORMATCMD        CEE3 GETRESULTS
 CA47 GOB1          ?C9E6 GRABSTATUS       4B GRP7CTR          CD49 GSERROR5
 CBE8 GTBOB         ?  51 HOSTID         CD33 ICBT15           CD2D ICBT5
 CF84 ICK1             01 IIC              05 INITCMD           27 IOERROR
 C080 IWM              07 IWMMODE        C08C L6CLR            C08D L6SET
 C08E L7CLR          C08F L7SET         ?  68 LASTONE         CBCF LASTPASS
   00 LOC0           ?  01 LOC1         ?CD36 LSTBSYWAIT5     C086 LSTRBCLR
 C087 LSTRBSET        C9E3 MARKERR         CE18 MAYBECTRL      CFD7 MDEV1
 CFD0 MDEV2           C50D MLIENTRY        CB5B MOD7TAB        C088 MONCLR
 C089 MONSET          C554 MORCHRS         CFBC MORDEVICES     07F8 MSLOT
   4D NEXT1             4E NEXT2             50 NEXT4            4D NEXT
   4F NEXT3             51 NEXT5             52 NEXT6            53 NEXT7
 CDFF NIN1              01 NOANSWER        CB7B NOAUXPTR       CEB1 NOCONTROL
   28 NODRIVE        CDC1 NOEH            CF0D NOERROR      ?  1F NOINT
?  02 NOMARK            20 NOPACKEND       CD6E NOPLAY         CDF1 NOTINIT
 CE72 NOTSTAT         CEBB NOWBLOCK        CED0 NOXTRASEND     CD43 NPENDERR5
 06F9 NUMDEVICES        4C ODDBYTES          2F OFFLINE       ?CDC7 OKAYCNT
 CA7C ONEMS1          CA7A ONEMS             C3 PACKETBEG        C8 PACKETEND
 CF86 PARMCTAB        C9BE PATCH1        ?  A5 PBBVALUE      ?  FF PBCVALUE
   00 PCID2             BF PDIDBYTE        CB4F PDIV7TAB       CB52 PMOD7TAB
?  52 POINTER           00 POWERRESET      C9D6 PREAMBLE     ?CBB0 PRECHECK
 C50A PRODOSENTRY     0473 PROFLAG         CD75 PZP            0BB8 RC1
   05 RC2             C583 RCODE             4B RCVBUF         CC50 RCVCOUNT
 C9F8 RDH1            CA02 RDH2            CA10 RDH3           CD39 RDH45
?CA0E RDH5            CD1E RDHA5             01 READCMD        C9E6 RECEIVEPACK
 CB33 RECPACK         C080 REQCLR          C081 REQSET         CA60 RESETCHAIN
 C576 RESET           0573 RETRY2          04F3 RETRY            01 ROM
?  4F RPACKETTYPE     CB3A RPK1            CB4E RPOUT          C578 RST1
 CF60 RZP             CF31 SA2             CB76 SAP1        ?  00 SCDEVICESTAT
?  01 SCGETDCB      ?  03 SCGETDEVINFO     0473 SCHOLES        C99D SCM1
?  02 SCRETNLSTAT     C9CF SD10            C9B2 SD7            C9C9 SD9
 CAEB SDOUBT          CEBF SECONDSEND      CA51 SEND80         CA53 SENDBYTE
 CADD SENDDATA        C886 SENDONEPACK     CAEC SENDPACK       CB00 SENDPILE
```

```
 CC1F SETIWMODE     ?FE89 SETKBD       ?FE93 SETVID        CA9A SETXN0
 CA9D SHIFT1         CAAD SHIFT2        CABD SHIFT3        CACD SHIFT4
 0573 SHTEMP1        05F3 SHTEMPX       0673 SHTEMPY       C927 SKIP1
 C929 SKIP2          C963 SKIP3         C965 SKIP4         CE3F SKIPCOPY
   58 SLOT           CC72 SLOTDEPRD     C8E8 SOB1          C8F9 SOB2
 C900 SOB3        ?  67 SOFTERROR        40 SOFT          CB0D SPILE1
 CB32 SPILOUT        CFF2 SQUIRREL      C8AF SSB           C8B2 SSD
?0100 STACK          CA34 START0        CA3C START1       ?CC72 START25
 C903 START          CA4E START2        CC7C START35         4D STATBYTE
?  81 STATMARK         1E STATMTO     ?  00 STATUSCMD      CF30 STOREAWAY
 CC0B SUN1           CC02 SUN2         ?CC00 SUN           CC1E SUN3
 0778 SVBCH          06F8 SVBCL           10 SVMASK1       C797 SWPROTO
 C784 SWRTS2        ?C9D7 SYNCTAB       CC66 T71           CC6F T72
   59 TBODD            59 TEMP           60 THEOFF       ?C000 THEORG
 CC59 TIMES7           41 TOPBITS       C899 UBSY1           5A UNIT
?1000 VERSION       ?FC22 VTAB         CC35 WAITIWMOFF     CF45 WAITOFF
?  04 WASRESET      ?C9E2 WASTE12      C9E1 WASTE14       ?C9E0 WASTE16
?C9DF WASTE18       ?C9DC WASTE32      CC3E WIWM1          CC4B WIWM2
   5B WPACKETTYPE      02 WRITECMD     CB64 WRITEPREP        2B WRITEPROT
 CBBC XOR1          ?CBB9 XOR2         CBD5 XOR3          CBDC XOR4
 CBCD XOR5           CA73 YMSWAIT        40 ZEROPAGE        1C ZPSIZE
?CFFC ZZZZZ
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 30-APR-85 22:46
** TOTAL LINES ASSEMBLED  3969
** FREE SPACE PAGE COUNT   70
```