

```

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: 0101 1 version equ $101 ;v1.0.1
0000: 0101 2 X6502
0000: 3 MSB ON
0000: 4 1st on, vsym, asym

```

01 PC

Protocol Converter Code for Tiger 20-OCT-86 06:29 PAGE 2

```

0000: 7 ; PPPP RRRR 000 TTTT 000 CCC 000 L
0000: 8 ; P P R R O O T O O C C O O L
0000: 9 ; PPPP RRRR O O T O O C O O L
0000: 10 ; P R R O O T O O C C O O L
0000: 11 ; P R R 000 T 000 CCC 000 LLLLL
0000: 12 ;
0000: 13 ; CCC 000 N N V V EEEEE RRRR TTTT EEEEE RRRR
0000: 14 ; C C O O N N N V V E R R T E R R
0000: 15 ; C O O N N N V V EEEEE RRRR T EEEEE RRRR
0000: 16 ; C C O O N N N V V E R R T E R R
0000: 17 ; CCC 000 N N V EEEEE R R T EEEEE R R

```

```

0000: 19 *****
0000: 20 *
0000: 21 * UniDisk 3.5 Driver Firmware Version 1.0.1
0000: 22 *
0000: 23 * Written by Michael Askins May 15, 1985
0000: 24 * Revised by M. Askins and R. Chiang April 10, 1986
0000: 25 *
0000: 26 * Copyright Apple Computer, Inc. 1985,1986
0000: 27 * All Rights Reserved
0000: 28 *
0000: 29 *****

```

01 PC Protocol Converter Code for Tiger 20-OCT-86 06:29 PAGE 3

```

0000: 31 *****
0000: 32 *
0000: 33 * Modification History:
0000: 34 *
0000: 35 * Rel Date Who Action
0000: 36 * -----
0000: 37 * *** 18 Dec 84 MSA RELEASE VERSION 0.02 (Sony)
0000: 38 * 10 Jan 85 MSA Added //c support:
0000: 39 * General conditional assembly overhead
0000: 40 * 16 Jan 85 MSA Added retries and timeouts
0000: 41 * Mslot handled correctly
0000: 42 * Finished Boot code
0000: 43 * Altered ProDOS errors - add $27 catchall
0000: 44 * 18 Jan 85 MSA Remove call to WAIT in monitor
0000: 45 * Add Boot failure messages
0000: 46 * 22 Jan 85 MSA Add IWM reconfigure for //c version
0000: 47 * 23 Jan 85 MSA Move Comm routines to $C800 ($C900)
0000: 48 * Fixed zero page preservation
0000: 49 * *** 23 Jan 85 MSA RELEASE VERSION 0.03 (Apple)
0000: 50 * 25 Jan 85 MSA Swap slot dep read and boot code (//c)
0000: 51 * Add other //c differences...
0000: 52 * 30 Jan 85 MSA Add auxtype byte
0000: 53 * Fix comm error on receive packet
0000: 54 * Fix checksum to include MSBs of overhead
0000: 55 * 07 Feb 85 MSA Add COUT support on boot fail
0000: 56 * *** 08 Feb 85 MSA RELEASE VERSION 1.00A (alpha)
0000: 57 * 22 Feb 85 MSA Add bytecount in X,Y on PC calls
0000: 58 * Change hard reset time to 1 ms (was 83)
0000: 59 * Crunched code by adding CtrPhases
0000: 60 * Add zeroing of third block byte (ProDOS)
0000: 61 * 06 Mar 85 MSA Fixed slot 7 goof (stack screw up)
0000: 62 * No clear phases on retries
0000: 63 * Hard reset time to 40 ms
0000: 64 * Pass #parms instead of unit# and no chk
0000: 65 * Init code (all reset vs. comm reset)
0000: 66 * Add 2 bytes to pass a full 9 byte cmd
0000: 67 * 16 Mar 85 MSA Fix bytecount on retries
0000: 68 * Boot block must be $800-$01, $801<-$00
0000: 69 * 17 Mar 85 MSA Remove WRREQ while waiting for motor TO
0000: 70 * Remove glitch on /ENBL2 in AssignID
0000: 71 * 20 Mar 85 MSA Add interrupt on/off/poll support
0000: 72 * Reset pulse to 80 ms
0000: 73 * //c delay of 100 ms on initial AssignID
0000: 74 * ID bytes changed
0000: 75 * Retransmit implemented (RecPack)
0000: 76 * Add send data packet retries (5)
0000: 77 * Rearrange PC stack adjust
0000: 78 * Add //c Appletalk vector
0000: 79 * 24 Mar 85 MSA Add //c millisecond wait each call
0000: 80 * *** 25 Mar 85 MSA RELEASE VERSION 1.00B (beta) (//e)
0000: 81 * 18 Apr 85 MSA Clear decimal mode
0000: 82 * Eight bytes are returned on stat unit#0
0000: 83 * Stat Unit#0 scode>0 is rejected
0000: 84 * X and Y set to 0008 on status unit#0
0000: 85 * Enable interrupts done correctly
0000: 86 * Add unit#0 parameter count checking
0000: 87 * *** 22 Apr 85 MSA RELEASE VERSION 1.01B
0000: 88 * *** 15 May 85 MSA RELEASE VERSION 1.0

```

01 PC Protocol Converter Code for Tiger 20-OCT-86 06:29 PAGE 4

```

0000: 89 * 10 Apr 86 RC removed reference to AppleTalk
0000: 90 * 10 Apr 86 RC forgot to add hi byte of auxptr in
0000: 91 * divide7 routine
0000: 92 * 10 Apr 86 RC returns write protect from old Lirons
0000: 93 * *** 10 Apr 86 RC RELEASE VERSION 1.0.1
0000: 94 * RC v1.0.1 is to be used with Tiger only
0000: 95 *
0000: 96 *****

```

----- NEXT OBJECT FILE NAME IS PC.0

```

C500: C500 98 org $C500
C500: 99 include pc.equates

```

```

C500:      2 *
C500:      00BF 3 PDIDByte equ $BF ;ProDOS attributes byte
C500:      0000 4 PCID2 equ $0 ;This means a Liron card
C500:      5 *
C500:      6 *
C500:      7 *
C500:      8 * Zero Page (temps) *
C500:      9 *
C500:     10 *
C500:     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 CMDBuffer1 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 bytecount1 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 *
C500:     64 dend
C500:     65 *
C500:     CFFF 66 ClearIORMs equ $CFFF
C500:     0100 67 stack equ $100
C500:     68 *
C500:     69 *
C500:     70 *
C500:     71 *
C500:     72 * Screenhole Storage *
C500:     73 *
C500:     74 *
C500:     75 *
C500:     76 * The screenhole layout is as follows:
C500:     77 *
C500:     78 * //e //c
C500:     79 *
C500:     80 * ProFlag $478+n $478
C500:     81 * Retry $4F8+n $4F8
C500:     82 * SHTemp1 $578+n $578
C500:     83 * SHTempX $5F8+n $5F8
C500:     84 * SHTempY $678+n $678
C500:     85 * Power1 $6F8+n ---
C500:     86 * Power2 $778+n ---
C500:     87 * NumDevices $7F8+n $6FE
C500:     88 * SvBcL $6F8 $6F8
C500:     89 * SvBcH $778 $778
C500:     90 *
C500:     0473 91 scholes equ $473 ;Use the slot 0 scholes for temps
C500:     92 *
C500:     0473 93 ProFlag equ scholes
C500:     04F3 94 Retry equ scholes+$80
C500:     0573 95 SHTemp1 equ scholes+$100
C500:     0573 96 Retry2 equ SHTemp1
C500:     05F3 97 SHTempX equ scholes+$180
C500:     0673 98 SHTempY equ scholes+$200
C500:     06F9 99 NumDevices equ $6F9 ;Actually in slot 6
C500:     100 *
C500:     06F8 101 SvBcL equ $6F8
C500:     0778 102 SvBcH equ $778
C500:     103 *
C500:     0025 104 cv equ $25
C500:     0024 105 ch equ $24
C500:     FC22 106 vtab equ $FC22
C500:     FD0D 107 cout equ $FD0D
C500:     07DB 108 bootscrn equ $7DB
C500:     07F8 109 MSlot equ $7F8
C500:     FE93 110 setvid equ $FE93
C500:     FE89 111 setkbd equ $FE89
C500:     FABA 112 AutoScan equ $FABA
C500:     E000 113 Basic equ $E000
C500:     0000 114 loc0 equ $0 ;Boot parms
C500:     0001 115 loc1 equ $1 ;
C500:     116 *
C500:     C797 117 SWPROTO equ $C797 ;//c bank switch to $C800

```


02 PC.EQUATES Equates

20-OCT-86 06:29 PAGE 7

```

C500: C784 118 SWRTS2 equ $C784 ;RTS to bank 1
C500: 119 *
C500: 120 *
C500: 121 *****
C500: 122 *
C500: 123 * General Equates *
C500: 124 *
C500: 125 *****
C500: 126 *
C500: 00A5 127 PBBValue equ $A5 ;Powerup Byte Base Value
C500: 00FF 128 PBCValue equ $FF ;Powerup Byte Complement Value
C500: 129 *
C500: 0000 130 PowerReset equ $00
C500: 0080 131 CommReset equ $80
C500: 132 *
C500: 0032 133 bsytl equ 50 ;(.55 ms) T/O on /BSY before send
C500: 000A 134 bsyto2 equ 10 ;(.12 ms) T/O on /BSY after send
C500: 001E 135 statnto equ 30 ;30 bytes stat mark timeout
C500: 0009 136 cmdlength equ 9 ;Command packet length
C500: 00C3 137 packetbeg equ $C3 ;Mark at beginning of packet
C500: 00C8 138 packetend equ $C8 ;End of packet mark
C500: 0080 139 cmdmark equ $80 ;Command packet identifier
C500: 0081 140 statmark equ $81 ;Status Packet identifier
C500: 0082 141 datamark equ $82 ;Data Packet identifier
C500: 142 *
C500: 0007 143 iwmmode equ $07 ;No timer, asynch, latch
C500: 144 *
C500: 0000 145 SCDeviceStat equ 0 ;Get Device Specific Status
C500: 0001 146 SCGetDCB equ 1 ;Get Dev Ctrl Block (modebits)
C500: 0002 147 SCRetNLStat equ 2 ;Return Newline Status
C500: 0003 148 SCGetDevInfo equ 3 ;Get Device Info Block
C500: 149 *
C500: C080 150 iwm equ $C080
C500: 151 *
C500: C080 152 reqclr equ iwm+0
C500: C081 153 reqset equ iwm+1
C500: C082 154 calclr equ iwm+2
C500: C083 155 calset equ iwm+3
C500: C084 156 ca2clr equ iwm+4
C500: C085 157 ca2set equ iwm+5
C500: C086 158 lstrbclr equ iwm+6
C500: C087 159 lstrbset equ iwm+7
C500: C088 160 monclr equ iwm+8
C500: C089 161 monset equ iwm+9
C500: C08A 162 enable1 equ iwm+10
C500: C08B 163 enable2 equ iwm+11
C500: C08C 164 l6clr equ iwm+12
C500: C08D 165 l6set equ iwm+13
C500: C08E 166 l7clr equ iwm+14
C500: C08F 167 l7set equ iwm+15
C500: 168 *
C500: 169 *
C500: 170 * Error0 codes
C500: 171 *
C500: 0001 172 noanswer equ 1
C500: 0002 173 nemark equ 2
C500: 0004 174 wasreset equ 4
C500: 0008 175 bytecmp equ 8

```

02 PC.EQUATES Equates

20-OCT-86 06:29 PAGE 8

```

C500: 0010 176 csumerr equ $10
C500: 0020 177 nopackend equ $20
C500: 0040 178 bushog equ $40
C500: 179 *
C500: 180 * Command Codes
C500: 181 *
C500: 0000 182 StatusCmd equ $00
C500: 0001 183 ReadCmd equ $01
C500: 0002 184 WriteCmd equ $02
C500: 0003 185 FormatCmd equ $03
C500: 0004 186 ControlCmd equ $04
C500: 0005 187 InitCmd equ $05
C500: 188 *
C500: 189 *
C500: 0040 190 Soft equ $01000000 ;The soft error bit in statbyte
C500: 191 *
C500: 0001 192 BadCmd equ $01
C500: 0004 193 BadPCnt equ $04
C500: 0006 194 BusErr equ $06
C500: 0011 195 BadUnit equ $11
C500: 001F 196 NoInt equ $1F
C500: 0021 197 BadCtl equ $21
C500: 0022 198 BadCtlParm equ $22
C500: 0027 199 IOError equ $27
C500: 0028 200 NoDrive equ $28
C500: 002B 201 WriteProt equ $2B
C500: 002D 202 BadBlock equ $2D
C500: 002F 203 OffLine equ $2F
C500: 0068 204 LastOne equ Soft+NoDrive
C500: 0067 205 SoftError equ Soft+IOError
C500: 206 *
C500: 0010 207 SWMask1 equ $10
C500: 208 *
C500: 08B8 209 RC1 equ 3000 ;Send a command pack 3000 times (3 sec)
C500: 0005 210 RC2 equ 5 ;Data Packs get tried only 5 times
C500: 211 *
C500: 212 *
C500: 100 include pc.bootospace

```

```

C500: 0060 2 TheOff equ $60 ;On //c IMM in slot 6
C500: 3 *
C500: 4 * Here beginneth that code which resideth in the boot space
C500: 5 * at the time the card resteth in slot the fifth.
C500: 6 *
C500: C500 7 C500org equ *
C500: 8 *
C500: 9 * Auto Boot signature bytes
C500: 10 * This is also the boot (auto & PR#5) entry point.
C500: 11 *
C500:A2 20 12 ldx #$20
C502:A2 00 13 ldx #$00
C504:A2 03 14 ldx #$03
C506: 15 *
C506:C9 00 16 cmp #0 ;Flag that this is a boot
C508:B0 17 C521 bcs BootC
C50A: 18 *
C50A: 19 * Here is the ProDOS normal entry point
C50A: 20 *
C50A: C50A 21 ProDOSEntry equ *
C50A: 22 *
C50A: 23 * Set up so that ProFLAG will have the top bit set
C50A: 24 *
C50A:38 25 sec
C50B:B0 01 C50E 26 bcs *+3 ;Skip the clear
C50D: 27 *
C50D: 28 * This is the MLJxface entry point
C50D: 29 *
C50D: C50D 30 MLJEntry equ * ;Only use this label in //c version
C50D:18 31 clc
C50E:A2 05 32 ldx #$05
C510:7E 73 04 33 ror ProFLAG,x ;ProFLAG[7]=1 if ProDOS, =0 if MLJ
C513:18 34 clc ;This is not a boot entry
C514: 35 *
C514: 36 * Now save mslot and clear all $C800 ROMs
C514: 37 *
C514: C514 38 bootcase5 equ *
C514:A2 C5 39 ldx #$C5 ;load value for MSLot
C516:8E F8 07 40 stx MSLot
C519:A2 05 41 ldx #$05
C51B:AD FF CF 42 lda ClearIOROMs ;Clear all $C800 latches but ours
C51E: 43 *
C51E:4C 97 C7 44 jmp SWPROTO
C521: C521 45 BootC equ *
C521:A2 05 46 ldx #$05 ;Need slot number
C523: 101 include pc.boot

```

```

C523: 2 *
C523: C523 3 Bootcode equ *
C523:86 58 4 stx slot
C525: 5 *
C525:A9 C5 6 lda #$C5
C527:8D F8 07 7 sta MSLot
C52A:20 76 C5 8 jsr reset
C52D: 9 *
C52D:A0 05 10 ldy #5 ;Copy a command table
C52F:B9 70 C5 11 bcl lda boottab,y
C532:99 42 00 12 sta cmdcode,y
C535:88 13 dey
C536:10 F7 C52F 14 bpl bcl
C538: 15 *
C538: 16 * Now do the read from block zero
C538: 17 *
C538:20 0A C5 18 jsr ProDOSEntry
C53B:B0 15 C552 19 bcs bootfail ;If fail, check loc
C53D: 20 *
C53D:AE 00 08 21 ldx $800 ;If ($800)<>1 this is no A// boot disk
C540:CA 22 dex
C541:D0 0F C552 23 bne bootfail
C543: 24 *
C543:AE 01 08 25 ldx $801 ;If $801 is zero, no boot
C546:F0 0A C552 26 beq bootfail
C548: 27 *
C548: 28 * It all looks okay. Jump to the code with NO in X.
C548: 29 *
C548:A5 58 30 lda Slot
C54A:0A 31 asl a
C54B:0A 32 asl a
C54C:0A 33 asl a
C54D:0A 34 asl a
C54E:AA 35 tax
C54F:4C 01 08 36 jmp $801 ;Jump to it
C552: 37 *
C552: 38 * Do this code if the boot can't be done.
C552: 39 * If this was an autoboot (loc=$CN00), continue the slot scan.
C552: 40 * If not, drop into basic after issuing appropriate message
C552: 41 *
C552:A2 10 42 bootfail ldx #>bmsglen-1
C554:BD 5F C5 44 morchr lda bootmsg,x
C557:9D DB 07 45 sta bootscrn,x
C55A:CA 46 dex
C55B:10 F7 C554 47 bpl morchr
C55D:80 FE C55D 48 coma bra coma ;He's dead Jim.
C55F:C3 E8 E5 E3 50 bootmsg asc 'Check Disk Drive.'
C570: 0011 51 bmsglen equ *-bootmsg
C570: 52 *
C570:01 50 00 08 53 boottab dfb ReadCMD,$50,0,8,0,0 ;Read from 1st; blk0->$801
C576: 54 *
C576: 55 *
C576: 56 * this routine is called from the //c reset code. it forces a
C576: 57 * reset of the PC Bus. location 0 and 1 are being used by the
C576: 58 * autostart people.

```


04 PC.Boot Service Boot Request 20-OCT-86 06:29 PAGE 11

```

C576:      C576 60 Reset   equ  *
C576:A2 08    61       ldx  #8
C578:      C578 62 rstl   equ  *
C578:BD 83 C5 63       lda  rcode,x
C578:95 02    64       sta  loc0+2,x
C57D:CA      65       dex
C57E:10 F8 C578 66      bpl  rstl
C580:4C 02 00 67       jmp  loc0+2

C583:      C583 69 rcode   equ  *
C583:20 0D C5 70       jsr  MLIEntry
C586:05      71       dfb  InitCMD
C587:09 00    72       dw  $0009
C589:60      73       rts

C58A:01 00    75 cmdlist dfb  1,0      ;One parm - the unit $00

---- NEXT OBJECT FILE NAME IS PC.1
C5F5:      C5F5 103      org  $C5F5
C5F5:4C 52 C5 104      jmp  bootfail ;Jump to the boot failure message
C5F8:4C 76 C5 105      jmp  reset   ;Reset vector
C5FB:00      106      dfb  PCID2
C5FC:00 00    107      dw  0
C5FE:BF      108      dfb  PDIDByte
C5FF:DA      109      dfb  >ProDOSEntry

---- NEXT OBJECT FILE NAME IS PC.2
C880:      C880 111      org  $C880
C880:4C 4C CD 112      jmp  Entry    ;The //c bank switch jumps here

C883:      114      include pc.packet
C883:      1       lst  cyc
C883:      2 *

```

05 PC.PACKET Send a CBus Packet 20-OCT-86 06:29 PAGE 12

```

C883:      4 *****
C883:      5 *
C883:      6 * SendOnePack          Send a CBus Packet *
C883:      7 *
C883:      8 * This routine sends a packet of data across the *
C883:      9 * bus. The protocol is as follows: *
C883:     10 *
C883:     11 * REQ _____|2_____5|_____ *
C883:     12 *
C883:     13 * /BSY _____|1_____3_____4|_____ *
C883:     14 *
C883:     15 * 1) Device signals ready for data *
C883:     16 * 2) Host signals data imminent *
C883:     17 * 3) Packet is transmitted (sync, command mark, *
C883:     18 *    ids, contents, checksum [msb=1]) *
C883:     19 * 4) Device signals packet recieved *
C883:     20 * 5) Host finishes send data cycle *
C883:     21 *
C883:     22 * The bytes are sent in slow mode (32 cycles/byte) *
C883:     23 * and the timing is critical. Branches which should *
C883:     24 * not cross page boundaries are marked. *
C883:     25 *
C883:     26 * Input: buffer (2 bytes) <- ptr to data to send *
C883:     27 *      bytecount (2) <- length (bytes) of data *
C883:     28 *      packettype (1) <- command or data packet *
C883:     29 *      CMDUnit (1) <- # of device to receive *
C883:     30 *
C883:     31 * Output: carry set- handshake error *
C883:     32 *      clr- bytes sent *
C883:     33 *
C883:     34 *****
C883:     35 *
C883:     C883 36 SendOnePack equ *
C883:     37 *
C883:     38 * Prep for the transmission *
C883:     39 *
C883:20 61 CB (6) 40      jsr  WritePrep ;Does a bunch of stuff
C886:      41 *
C886:      42 * Enable PC chain. *
C886:      43 *
C886:20 7D CA (6) 44      jsr  enablechain ;This sets X reg
C889:A0 07 (2) 45      ldy  #lwmmode ;This is the mode value
C88B:20 20 CC (6) 46      jsr  SetINMode ;Don't mess unless we gotta
C88E:      47 *
C88E:      48 * Turn on the INM *
C88E:      49 *
C88E:BD 8B C0 (4) 50      lda  enable2,x ;Don't disturb //c internal drive
C891:BD 89 C0 (4) 51      lda  monset,x
C894:      52 *
C894:      53 * Loop until the chain becomes unbusy *
C894:      54 *
C894:A0 32 (2) 55      ldy  #bsytol ;Each loop is 11 microseconds
C896:BD 8E C0 (4) 56      bsyl  lda  l7clr,x ;Test if /BSY is hi or lo
C899:30 07 C8A2(3) 57      bmi  chainunhsy ;If hi, bus is not busy
C89B:88 (2) 58      dey
C89C:D0 F8 C896(3) 59      bne  bsyl ;Keep trying
C89E:      60 *
C89E:38 (2) 61      sec

```

```

C89F:4C CC C9 (3) 62 jmp sd10
C8A2: 63 *
C8A2: 64 * Tell the bus that data is coming and send the sync bytes
C8A2: 65 * Sync is groups of eight 2's separated by a 6 (micS cell)
C8A2: 66 * (1111111001111111001111111100 ...)
C8A2: 67 *
C8A2: C8A2 68 chainunbsy equ *
C8A2:BD 81 C0 (4) 69 lda reqset,x ;Raise REQ
C8A5: 70 *
C8A5:A0 05 (2) 71 ldy #5 ;Sync plus packet begin
C8A7: 72 *
C8A7:A9 FF (2) 73 lda #5FF ;Send out the 1st byte sync
C8A9:9D 8F C0 (5) 74 sta l7set,x
C8AC: 75 *
C8AC:B9 D3 C9 (4) 76 ssb lda preamble,y
C8AF: 77 *
C8AF: 78 *
C8AF:1E 8C C0 (7) 79 ssd asl l6clr,x ;Wait 'til buffer empty
C8B2:90 FB C8AF(3) 80 bcc ssd
C8B4: 81 *
C8B4:9D 8D C0 (5) 82 sta l6set,x
C8B7:88 (2) 83 dey
C8B8:10 F2 C8AC(3) 84 bpl ssb ;Back for more bytes
C8BA: 85 *
C8BA: 86 * Send over the destination ID
C8BA: 87 *
C8BA:A5 5A (3) 88 lda Unit
C8BC:09 80 (2) 89 ora #80 ;Make the device ID
C8BE:20 50 CA (6) 90 jsr sendbyte
C8C1: 91 *
C8C1: 92 * Send the source ID (that's us... we're an $80)
C8C1: 93 *
C8C1:20 4E CA (6) 94 jsr send80
C8C4: 95 *
C8C4: 96 * Send over the packet type (command or data)
C8C4: 97 *
C8C4:A5 5B (3) 98 lda Wpackettype
C8C6:20 50 CA (6) 99 jsr sendbyte
C8C9: 100 *
C8C9: 101 * Send the Auxilliary Type byte (an $80 from this rev PC)
C8C9: 102 *
C8C9:20 4E CA (6) 103 jsr send80
C8CC: 104 *
C8CC: 105 * Send the status byte (null for us), and length bytes
C8CC: 106 *
C8CC:20 4E CA (6) 107 jsr send80
C8CF:A5 4C (3) 108 lda oddbytes
C8D1:09 80 (2) 109 ora #80
C8D3:20 50 CA (6) 110 jsr sendbyte
C8D6:A5 4B (3) 111 lda grp7ctr
C8D8:09 80 (2) 112 ora #80
C8DA:20 50 CA (6) 113 jsr sendbyte
C8DD: 114 *
C8DD: 115 * Now send the "oddbytes" part of the packet contents
C8DD: 116 *
C8DD:A5 4C (3) 117 lda oddbytes ;Get # of "odd" bytes
C8DF:F0 15 C8F6(3) 118 beq sob2 ;Skip if no odd bytes
C8E1: 119 *

```

```

C8E1:A0 FF (2) 120 ldy #5FF
C8E3:A5 59 (3) 121 lda tbodd ;Get the odd bytes msh's (A[7]=1)
C8E5: 122 *
C8E5:1E 8C C0 (7) 123 sob1 asl l6clr,x ;Do a write handshake
C8E8:90 FB C8E5(3) 124 bcc sob1
C8EA:9D 8D C0 (5) 125 sta l6set,x
C8ED:C8 (2) 126 iny
C8EE:B1 54 (5) 127 lda (buffer),y ;Get the data byte
C8F0:09 80 (2) 128 ora #80 ;Flip on the hi bit
C8F2:C4 4C (3) 129 cpy oddbytes ;Are we done?
C8F4:90 EF C8E5(3) 130 blt sob1
C8F6: 131 *
C8F6: 132 * Now send over the groups of seven contents
C8F6: 133 * Currently assume there must be at least one group of 'em
C8F6: 134 *
C8F6: C8F6 135 sob2 equ *
C8F6:A5 4B (3) 136 lda grp7ctr ;Check if there are groups to send
C8F8:D0 03 C8FD(3) 137 bne sob3 ;=> At least one group
C8FA:4C 96 C9 (3) 138 jmp datdone ;Skip to send checksum
C8FD: 139 *
C8FD: C8FD 140 sob3 equ *
C8FD:EA (2) 141 nop ;Waste 2 cycles
C8FE:A0 00 (2) 142 ldy #0
C900:A5 41 (3) 143 start lda topbits
C902:9D 8D C0 (5) 144 sta l6set,x
C905: 145 *
C905: 146 * Send first byte
C905: 147 *
C905:A5 4D (3) 148 lda next1
C907:09 80 (2) 149 ora #80
C909:84 59 (3) 150 sty temp ;Swap Y for short handshake
C90B:BC 8C C0 (4) 151 achel ldy l6clr,x ;Wait 'til buffer ready
C90E:10 FB C90B(3) 152 bpl achel
C910:9D 8D C0 (5) 153 sta l6set,x ;Send the byte
C913:A4 59 (3) 154 ldy temp ;Get back Y
C915: 155 *
C915: 156 * Prep the next "1st" byte for next time
C915: 157 *
C915:B1 56 (5) 158 lda (buffer2),y
C917:85 4D (3) 159 sta next1
C919:0A (2) 160 asl a
C91A:26 41 (5) 161 rol topbits ;Store the top bit
C91C:C8 (2) 162 iny ;Next byte
C91D: 163 *
C91D: 164 * It's possible that we're at a page boundary now. If so, bump the
C91D: 165 * hi order part of the pointer.
C91D: 166 *
C91D:D0 05 C924(3) 167 bne skip1
C91F:E6 57 (5) 168 inc buffer2+1
C921:4C 26 C9 (3) 169 jmp skip2
C924:48 (3) 170 skip1 pha ;Equalize the cases
C925:68 (4) 171 pla
C926: 172 *
C926: 173 * Push us ahead by an additional 8 cycles for margin reasons
C926: 174 * Plus I gotta get the topbits MSB set somehow...
C926: 175 *
C926: C926 176 skip2 equ *
C926:A9 02 (2) 177 lda #00000010 ;Flip what will be MSB

```


05 PC.PACKET Send a CBus Packet 20-OCT-86 06:29 PAGE 15

```

C928:05 41 (3) 178 ora tophits
C92A:85 41 (3) 179 sta tophits
C92C: 180 *
C92C: 181 * Send the second byte
C92C: 182 *
C92C:A5 4E (3) 183 lda next2
C92E:09 80 (2) 184 ora $80
C930:9D 8D C0 (5) 185 sta l6set,x ;Send the byte
C933:B1 56 (5) 186 lda (buffer2),y
C935:85 4E (3) 187 sta next2
C937:0A (2) 188 asl a
C938:26 41 (5) 189 rol tophits ;Store the top bit
C93A:C8 (2) 190 iny ;Next byte
C93B: 191 *
C93B: 192 * Send the third byte
C93B: 193 *
C93B:A5 4F (3) 194 lda next3
C93D:09 80 (2) 195 ora $80
C93F:9D 8D C0 (5) 196 sta l6set,x ;Send the byte
C942:B1 56 (5) 197 lda (buffer2),y
C944:85 4F (3) 198 sta next3
C946:0A (2) 199 asl a
C947:26 41 (5) 200 rol tophits ;Store the top bit
C949:C8 (2) 201 iny ;Next byte
C94A: 202 *
C94A: 203 * Send the fourth byte
C94A: 204 *
C94A:A5 50 (3) 205 lda next4
C94C:09 80 (2) 206 ora $80
C94E:9D 8D C0 (5) 207 sta l6set,x ;Send the byte
C951:B1 56 (5) 208 lda (buffer2),y
C953:85 50 (3) 209 sta next4
C955:0A (2) 210 asl a
C956:26 41 (5) 211 rol tophits ;Store the top bit
C958:C8 (2) 212 iny ;Next byte
C959: 213 *
C959: 214 * After the first 256 bytes, we will cross pages here. If we did
C959: 215 * cross, bump the buffer pointer. If not, equalize the cases with
C959: 216 * seven cycles of time wasting.
C959: 217 *
C959:D0 85 I960(3) 218 bne skip3
C95B:E6 57 (5) 219 inc buffer2+1
C95D:4C 62 C9 (3) 220 jmp skip4
C960:48 (3) 221 skip3 pha
C961:68 (4) 222 pla
C962: I962 223 skip4 equ *
C962: 224 *
C962: 225 * Send the fifth byte
C962: 226 *
C962:A5 51 (3) 227 lda next5
C964:09 80 (2) 228 ora $80
C966:9D 8D C0 (5) 229 sta l6set,x ;Send the byte
C969:B1 56 (5) 230 lda (buffer2),y
C96B:85 51 (3) 231 sta next5
C96D:0A (2) 232 asl a
C96E:26 41 (5) 233 rol tophits ;Store the top bit
C970:C8 (2) 234 iny ;Next byte
C971: 235 *

```

05 PC.PACKET Send a CBus Packet 20-OCT-86 06:29 PAGE 16

```

C971: 236 * Send the sixth byte
C971: 237 *
C971:A5 52 (3) 238 lda next6
C973:09 80 (2) 239 ora $80
C975:9D 8D C0 (5) 240 sta l6set,x ;Send the byte
C978:B1 56 (5) 241 lda (buffer2),y
C97A:85 52 (3) 242 sta next6
C97C:0A (2) 243 asl a
C97D:26 41 (5) 244 rol tophits ;Store the top bit
C97F:C8 (2) 245 iny ;Next byte
C980: 246 *
C980: 247 * Send the last byte of the group
C980: 248 *
C980:A5 53 (3) 249 lda next7
C982:09 80 (2) 250 ora $80
C984:9D 8D C0 (5) 251 sta l6set,x ;Send the byte
C987:B1 56 (5) 252 lda (buffer2),y
C989:85 53 (3) 253 sta next7
C98B:0A (2) 254 asl a
C98C:26 41 (5) 255 rol tophits ;Store the top bit
C98E:C8 (2) 256 iny ;Next byte
C98F: 257 *
C98F: 258 * Now see if we have sent enough groups of seven
C98F: 259 *
C98F:C6 4B (5) 260 dec grp7ctr
C991:F0 03 C996(3) 261 beq datdone
C993: 262 *
C993: 263 * Otherwise, back to do more. Note it's too far for a branch.
C993: 264 *
C993:4C 00 C9 (3) 265 jmp start
C996: 266 *
C996: 267 * Whew! Now send the damn checksum as two FM bytes
C996: 268 *
C996: C996 269 datdone equ *
C996:A5 40 (3) 270 lda checksum ;c7 c6 c5 c4 c3 c2 c1 c0
C998:09 AA (2) 271 ora $SAA ; 1 c6 1 c4 1 c2 1 c0
C99A:BC 8C C0 (4) 272 scml ldy l6clr,x
C99D:10 FB C99A(3) 273 bpl scml ;Handshake this byte
C99F:9D 8D C0 (5) 274 sta l6set,x ;These are even bits
C9A2: 275 *
C9A2:A5 40 (3) 276 lda checksum ;c7 c6 c5 c4 c3 c2 c1 c0
C9A4:4A (2) 277 lsr a ; 0 c7 c6 c5 c4 c3 c2 c1
C9A5:09 AA (2) 278 ora $SAA ; 1 c7 1 c5 1 c3 1 c1
C9A7:20 50 CA (6) 279 jsr sendbyte
C9AA: 280 *
C9AA: 281 * Send the end of packet mark
C9AA: 282 *
C9AA:A9 C8 (2) 283 lda #packetend
C9AC:20 50 CA (6) 284 jsr sendbyte
C9AF: 285 *
C9AF: 286 * Wait until write underflow
C9AF: 287 *
C9AF:BD 8C C0 (4) 288 sd7 lda l6clr,x
C9B2:29 40 (2) 289 and $340
C9B4:D0 F9 C9AF(3) 290 bne sd7 ;Still writing data
C9B6: 291 *
C9B6:9D 8D C0 (5) 292 sta l6set,x ;Back to sense mode (dummy write)
C9B9: 293 *

```



```

C9B9:      294 * Now wait until the drive acknowledges receipt of the
C9B9:      295 * string or until timeout
C9B9:      296 *
C9B9:A0 0A      (2) 297 ldy #bsyto2 ;Load timeout to see bsy low
C9BB:88      (2) 298 patchl dey ;A little closer to an error
C9BC:D0 08 C9C6(3) 299 bne sd9 ;There's still time
C9BE:      300 *
C9BE:      301 * Too much time has elapsed. Drive didn't get string.
C9BE:      302 *
C9BE:A9 01      (2) 303 lda #noanswer ;Report error in comm error byte
C9C0:      304 dberror equ *
C9C0:20 97 CA      (6) 305 jsr SetXN0 ;For dberror entry
C9C3:38      (2) 306 sec ;Signal a problem
C9C4:B0 06 C9CC(3) 307 bcs sd10
C9C6:      308 *
C9C6:      309 * See if drive has acknowledged the bytes yet
C9C6:      310 *
C9C6:BD 8E C0      (4) 311 sd9 lda l7clr,x ;Wait 'til /BSY lo
C9C9:30 F0 C9BB(3) 312 bmi patchl
C9CB:      313 *
C9CB:      314 * Finish the sequence
C9CB:      315 *
C9CB:18      (2) 316 clc ;This is a normal exit
C9CC:BD 80 C0      (4) 317 sd10 lda reqclr,x ;Set REQ lo
C9CF:BD 8C C0      (4) 318 lda l6clr,x ;Back into read mode
C9D2:      319 *
C9D2:      320 * Pull back the bytecount in all cases
C9D2:      321 *
C9D2:60      (6) 322 rts
C9D3:      323 *
C9D3:      324 *
C9D3:      325 * This table, when sent in reverse order, provides a
C9D3:      326 * sync pattern used to synchronize the drive IMM with
C9D3:      327 * the data stream. The first byte (last sent) is the
C9D3:      328 * packet begin mark.
C9D3:      329 *
C9D3:C3      330 preamble dfb packetbeg
C9D4:FF FC F3 CF      331 synctab dfb $FF,$FC,$F3,$CF,$3F
C9D9:      332 *
C9D9:      333 *

```

```

C9D9:      335 *
C9D9:      336 * These routines are for wasting specific amounts of time
C9D9:      337 * This code segment should not cross page boundaries.
C9D9:      338 *
C9D9:20 DE C9      (6) 339 waste32 jsr wastel4
C9DC:EA      (2) 340 wastel8 nop
C9DD:EA      (2) 341 wastel6 nop
C9DE:EA      (2) 342 wastel4 nop
C9DF:60      (6) 343 wastel2 rts
C9E0:      344 *
C9E0:      345 *
C9E0:      346 markerr equ *
C9E0:4C C0 C9      (3) 347 jmp dberror

```

05 PC.PACKET

Receive a CBus Packet

20-OCT-86 06:29 PAGE 19

```

C9E3: 349 *****
C9E3: 350 *
C9E3: 351 * ReceivePack      Get a packet from bus resident *
C9E3: 352 *
C9E3: 353 *
C9E3: 354 * REQ _____|2_____5|_____
C9E3: 355 *
C9E3: 356 * /BSY _____|1_____3_____4|_____
C9E3: 357 *
C9E3: 358 * 1) Drive signals ready to send packet *
C9E3: 359 * 2) Host signals ready to receive data *
C9E3: 360 * 3) Packet is transmitted (sync, mark, IDs, data, *
C9E3: 361 *      checksum [msb=1]) *
C9E3: 362 * 4) Drive signals packet dispatched *
C9E3: 363 * 5) Host acknowledges receipt of packet *
C9E3: 364 *
C9E3: 365 * The bytes are sent in slow mode (32 cycles/byte) *
C9E3: 366 * and the timing is critical. Branches which should *
C9E3: 367 * not cross page boundaries are marked. *
C9E3: 368 *
C9E3: 369 * Input: buffer <- address where packet guts left *
C9E3: 370 *
C9E3: 371 * Output: carry set- handshake error *
C9E3: 372 *      clr- bytes recieved *
C9E3: 373 *      A <- error0 if carry set *
C9E3: 374 *
C9E3: 375 *****
C9E3: 376 *
C9E3: C9E3 377 grabstatus equ *
C9E3: C9E3 378 ReceivePack equ *
C9E3: 379 *
C9E3: 380 * Init the checksum
C9E3: 381 *
C9E3:A9 00 (2) 382      lda    #000
C9E5:85 40 (3) 383      sta    checksum
C9E7: 384 *
C9E7: 385 * Copy over buffer -> buffer2
C9E7: 386 *
C9E7:A5 54 (3) 387      lda    buffer
C9E9:85 56 (3) 388      sta    buffer2
C9EB:A5 55 (3) 389      lda    buffer+1
C9ED:85 57 (3) 390      sta    buffer2+1
C9EF: 391 *
C9EF: 392 * Set up the indirect pointer for jump to 2nd part of code
C9EF: 393 *
C9EF:20 7D CA (6) 394      jsr    enablechain ;Set X register to $N0
C9F2: 395 *
C9F2:BD 8D C0 (4) 396      lda    l6set,x      ;Prep for sense mode
C9F5: 397 *
C9F5: 398 * Now wait for BSY to go hi, signalling 'ready w/ status'
C9F5: 399 *
C9F5:BD 8E C0 (4) 400 rdhl    lda    l7clr,x      ;Read sense
C9F8:10 FB C9F5(3) 401      bpl    rdhl      ;Wait til a high
C9FA: 402 *
C9FA: 403 * Signal Liron we're ready to receive
C9FA: 404 *
C9FA:BD 81 C0 (4) 405      lda    reqset,x      ;Raise /REQ
C9FD: 406 *

```

05 PC.PACKET

Receive a CBus Packet

20-OCT-86 06:29 PAGE 20

```

C9FD: 407 * Wait for a byte from Liron or timeout
C9FD: 408 *
C9FD:A0 1E (2) 409      ldy    #statmt0      ;Max bytes 'til stat mark
C9FF:BD 8C C0 (4) 410 rdh2    lda    l6clr,x
CA02:10 FB C9FF(3) 411      bpl    rdh2      ;*** No Page Cross ***
CA04:88 (2) 412      dey
CA05:30 D9 C9E0(3) 413      bmi    markerr      ;Didn't find a packet in time
CA07: 414 *
CA07: 415 * Is it the beginning of the packet?
CA07: 416 *
CA07:C9 C3 (2) 417      cmp    #packetbeg      ;Find the packet begin mark
CA09:D0 F4 C9FF(3) 418      bne    rdh2      ;Back again - no timeout for now
CA0B: 419 *
CA0B: 420 * Okay load up the table with this stuff
CA0B: 421 *
CA0B: CA0B 422 rdh5    equ    *
CA0B: 423 *
CA0B:A0 06 (2) 424      ldy    #6      ;Seven bytes of overhead
CA0D:BD 8C C0 (4) 425 rdh3    lda    l6clr,x      ;If byte ready, grab it
CA10:10 FB CA0D(3) 426      bpl    rdh3      ;*** No Page Cross ***
CA12:29 7F (2) 427      and    #01111111 ;Strip start bit
CA14:99 4B 00 (5) 428      sta    rcvbuf,y
CA17:49 80 (2) 429      eor    #580      ;Pop MSB back on for checksum
CA19:45 40 (3) 430      eor    checksum
CA1B:85 40 (3) 431      sta    checksum
CA1D:88 (2) 432      dey
CA1E:10 ED CA0D(3) 433      bpl    rdh3
CA20: 434 *
CA20: 435 * Set groups of seven buffer pointer buffer2
CA20: 436 *
CA20:A5 4C (3) 437      lda    oddbytes
CA22:F0 27 CA4B(3) 438      beq    start2      ;Skip alteration if no oddbytes
CA24:18 (2) 439      clc
CA25:65 54 (3) 440      adc    buffer
CA27:85 56 (3) 441      sta    buffer2
CA29:A5 55 (3) 442      lda    buffer+1
CA2B:69 00 (2) 443      adc    #0
CA2D:85 57 (3) 444      sta    buffer2+1
CA2F: 445 *
CA2F:A0 00 (2) 446      ldy    #0
CA31: 447 *
CA31: 448 * Now receive the odd bytes
CA31: 449 *
CA31:BD 8C C0 (4) 450 start0 lda    l6clr,x      ;Read in the odd bytes topbits
CA34:10 FB CA31(3) 451      bpl    start0
CA36:0A (2) 452      asl    a      ;Pop off the start bit
CA37:85 41 (3) 453      sta    topbits
CA39: CA39 454 start1 equ    *
CA39:BD 8C C0 (4) 455      lda    l6clr,x      ;Get an odd byte
CA3C:10 FB CA39(3) 456      bpl    start1
CA3E:06 41 (5) 457      asl    topbits      ;Get an MSB
CA40:B0 02 CA44(3) 458      bcs    gob1      ;If MSB set, leave start bit
CA42:49 80 (2) 459      eor    #580      ;MSB clear- flip start bit
CA44:91 54 (6) 460 gob1    sta    (buffer),y ;Squirrel it away
CA46:C8 (2) 461      iny      ;Next spot
CA47:C4 4C (3) 462      cpy    oddbytes      ;Are we done?
CA49:90 EE CA39(3) 463      bit    start1      ;If more, branch
CA4B: 464 *

```



```

CA4B:      CA4B      465 start2 equ *
CA4B:4C 73 CC      (3) 466      jmp SlotDepRd
CA4E:      467 *
CA4E:      CA4E      468 Send80 equ *
CA4E:A9 80      (2) 469      lda #80
CA50:      CA50      470 SendByte equ *
CA50:BC 8C C0      (4) 471      ldy 16clr,x
CA53:10 FB CA50(3) 472      bpl SendByte
CA55:9D 8D C0      (5) 473      sta 16set,x
CA58:45 40      (3) 474      eor checksum
CA5A:85 40      (3) 475      sta checksum
CA5C:60      (6) 476      rts
CA5D:      477 *
CA5D:      478 *
CA5D:      479 *
CA5D:      480 *
CA5D:      CA5D      481 resetchain equ *
CA5D:20 87 CA      (6) 482      jsr ClrPhases
CA60:BD 81 C0      (4) 483      lda reqset,x
CA63:BD 85 C0      (4) 484      lda ca2set,x
CA66:A0 50      (2) 485      ldy #80 ;Hard reset for 80 ms
CA68:20 70 CA      (6) 486      jsr YMSWait
CA6B:      487 *
CA6B:20 87 CA      (6) 488      jsr ClrPhases
CA6E:      489 *
CA6E:A0 0A      (2) 490      ldy #10 ;About 10 mS reset time!
CA70:      491 *
CA70:      CA70      492 YMSWait equ *
CA70:20 77 CA      (6) 493      jsr OneMS
CA73:88      (2) 494      dey
CA74:D0 FA CA70(3) 495      bne YMSWait
CA76:60      (6) 496      rts
CA77:      497 *
CA77:      CA77      498 OneMS equ *
CA77:A2 C8      (2) 499      ldx #200
CA79:CA      (2) 500 onemsl dex
CA7A:D0 FD CA79(3) 501      bne onemsl
CA7C:60      (6) 502      rts
CA7D:      503 *
CA7D:      504 *
CA7D:      CA7D      505 enablechain equ *
CA7D:20 97 CA      (6) 506      jsr SetXNO
CA80:BD 83 C0      (4) 507      lda calset,x
CA83:BD 87 C0      (4) 508      lda lstrbset,x
CA86:60      (6) 509      rts
CA87:      510 *
CA87:      511 *
CA87:      CA87      512 ClrPhases equ *
CA87:20 97 CA      (6) 513      jsr SetXNO
CA8A:BD 80 C0      (4) 514      lda reqclr,x
CA8D:BD 82 C0      (4) 515      lda calclr,x
CA90:BD 84 C0      (4) 516      lda ca2clr,x
CA93:BD 86 C0      (4) 517      lda lstrbclr,x
CA96:60      (6) 518      rts
CA97:      519 *
CA97:      520 *
CA97:      CA97      521 SetXNO equ *
CA97:A2 60      (2) 522      ldx #860

```

```

CA99:      523 *
CA99:60      (6) 524      rts
CA9A:      525 *
CA9A:      526 * Shift tables for use when reading. Each table should not
CA9A:      527 * straddle pages.
CA9A:      528 *
CA9A:80 80 80 80 529 shift1 dfb $80,$80,$80,$80,$80,$80,$80,$80
CAA2:00 00 00 00 530      dfb 0,0,0,0,0,0,0,0
CAA8:80 80 80 80 531 shift2 dfb $80,$80,$80,$80,0,0,0,0
CAB2:80 80 80 80 532      dfb $80,$80,$80,$80,0,0,0,0
CABA:80 80 00 00 533 shift3 dfb $80,$80,0,0,$80,$80,0,0
CAC2:80 80 00 00 534      dfb $80,$80,0,0,$80,$80,0,0
CACA:80 00 80 00 535 shift4 dfb $80,0,$80,0,$80,0,$80,0
CAD2:80 00 80 00 536      dfb $80,0,$80,0,$80,0,$80,0
CADA:      537 *
CADA:      538 *

```

05 PC.PACKET Receive a CBus Packet 20-OCT-86 06:29 PAGE 23

```

CADA:          540 *
CADA:          CADA 541 SendData equ *
CADA:A9 05     (2) 542 ldy #>RC2
CADC:A0 00     (2) 543 ldy #<RC2
CADE:20 FD CA (6) 544 jsr SendPile
CAE1:90 05     (3) 545 bcc sdoubt
CAE3:A9 80     (2) 546 lda #CommReset
CAE5:20 98 CF (6) 547 jsr AssignID
CAE8:          CAE8 548 sdoubt equ *
CAE8:60        (6) 549 rts
CAE9:          550 *
CAE9:          551 *
CAE9:          CAE9 552 SendPack equ *
CAE9:20 FD CA (6) 553 jsr SendPile ;Try to send a pack
CAEC:90 FA     (3) 554 bcc sdoubt
CAEE:A9 80     (2) 555 lda #CommReset ;This is a communications failure
CAF0:20 98 CF (6) 556 jsr AssignID ;Reset to try again
CAF3:          557 *
CAF3:AD F8 06 (4) 558 lda SvBcL ;Get back the packetlength
CAF6:85 4D     (3) 559 sta bytecountl
CAF8:AD 78 07 (4) 560 lda SvBcH
CAF8:85 4E     (3) 561 sta bytecounth
CAFD:          562 *
CAFD:          CAFD 563 SendPile equ *
CAFD:A9 B8     (2) 564 lda #>RC1 ;Retry count (big!)
CAFF:A0 08     (2) 565 ldy #<RC1
CB01:          566 *
CB01:          CB01 567 AltSendPile equ *
CB01:A6 58     (3) 568 ldx slot
CB03:9D F3 04 (5) 569 sta Retry,x
CB06:98        (2) 570 tya
CB07:9D 73 05 (5) 571 sta Retry2,x
CB0A:          572 *
CB0A:          573 * SendPack destroys the bytecount
CB0A:          574 *
CB0A:          CB0A 575 spilol equ *
CB0A:A5 4D     (3) 576 lda bytecountl
CB0C:8D F8 06 (4) 577 sta SvBcL
CB0F:A5 4E     (3) 578 lda bytecounth
CB11:8D 78 07 (4) 579 sta SvBcH
CB14:          580 *
CB14:20 83 C8 (6) 581 jsr SendOnePack ;Send the packet
CB17:          582 *
CB17:AD F8 06 (4) 583 lda SvBcL
CB1A:85 4D     (3) 584 sta bytecountl
CB1C:AD 78 07 (4) 585 lda SvBcH
CB1F:85 4E     (3) 586 sta bytecounth
CB21:          587 *
CB21:90 0C     (3) 588 bcc spilout
CB23:A6 58     (3) 589 ldx slot
CB25:DE F3 04 (7) 590 dec Retry,x
CB28:D0 E0     (3) 591 bne spilol
CB2A:DE 73 05 (7) 592 dec Retry2,x
CB2D:10 DB     (3) 593 bpl spilol ;If all fails, carry is set
CB2F:60        (6) 594 spilout rts
CB30:          595 *
CB30:          CB30 596 RecPack equ *
CB30:A4 58     (3) 597 ldy Slot

```

05 PC.PACKET Receive a CBus Packet 20-OCT-86 06:29 PAGE 24

```

CB32:A9 05     (2) 598 lda #>RC2
CB34:99 F3 04 (5) 599 sta Retry,y
CB37:          CB37 600 rpkl equ *
CB37:20 F3 C9 (6) 601 jsr ReceivePack
CB3A:90 0F     (3) 602 bcc rpout
CB3C:A0 01     (2) 603 ldy #1
CB3E:20 70 CA (6) 604 jsr YMSWait
CB41:20 C0 C9 (6) 605 jsr dberror ;Recycle handshake and set carry
CB44:A6 58     (3) 606 ldx Slot
CB46:DE F3 04 (7) 607 dec Retry,x
CB49:D0 EC     (3) 608 bne rpkl ;Carry set still
CB4B:          CB4B 609 rpout equ *
CB4B:60        (6) 610 rts
CB4C:          611 *
CB4C:          612 *

```

```

CB4C:          614 *
CB4C:          615 *

```

```

CB86:          672 * bits in the 10 order bytecount, correcting each time H00 becomes
CB86:          673 * bigger than 2.

```



```

CB4C:      614 *****
CB4C:      615 *
CB4C:      616 * Divide?          Do DIV and MOD 7 and set auxptr *
CB4C:      617 *
CB4C:      618 * This routine divides the bytecount by seven. The *
CB4C:      619 * quotient gives the number of groups of seven bytes to *
CB4C:      620 * be sent, and the remainder gives the number of "odd" *
CB4C:      621 * bytes. *
CB4C:      622 *
CB4C:      623 * Input:  bytecountl,h <- # of bytes to write *
CB4C:      624 * buffer    <- pointer to data *
CB4C:      625 * Output: auxptr    <- pointer to speed up csumming *
CB4C:      626 * oddbytes   <- bytecount MOD 7 *
CB4C:      627 * grp7ctr   <- bytecount DIV 7 *
CB4C:      628 *
CB4C:      629 *****
CB4C:      630 *
CB4C:00 24 49 631 pdiv7tab dfb 0,36,73
CB4F:00 04 01 632 pmod7tab dfb 0,4,1
CB52:00 01 02 04 633 div7tab dfb 0,1,2,4,9,18
CB58:00 01 02 04 634 mod7tab dfb 0,1,2,4,1,2
CB5E:      635 *
CB5E:00 7F FF 636 auxptrinc dfb 0,$7F,$FF
CB61:      637 *
CB61:      CB61 638 WritePrep equ *
CB61:      CB61 639 Divide7 equ *
CB61:      640 *
CB61:      641 * Set up auxptr <- buffer+$80 if $0FF < bytecount < $200
CB61:      642 * or auxptr <- buffer+$100 if $1FF < bytecount
CB61:      643 *
CB61:A6 4E (3) 644 ldx bytecountl ;0, 1 or 2
CB63:F0 17 CB7C(3) 645 beq noauxptr ;Auxptr used only for full pages
CB65:      646 *
CB65:A5 55 (3) 647 lda buffer+1
CB67:85 57 (3) 648 sta auxptr+1 ;Copy over hi order part
CB69:      649 *
CB69:A9 80 (2) 650 lda #$80 ;Anticipate smaller bytecount
CB6B:E0 01 (2) 651 cpx #1 ;Check bytecount
CB6D:F0 04 CB73(3) 652 beq sapl ;=> $0FF < bytecount < $200
CB6F:      653 *
CB6F:E6 57 (5) 654 inc auxptr+1 ;Add $100 to bytecount instead
CB71:A9 00 (2) 655 lda #0 ;Make sure lo order unaltered
CB73:18 (2) 656 sapl clc
CB74:65 54 (3) 657 adc buffer
CB76:85 56 (3) 658 sta auxptr
CB78:90 02 CB7C(3) 659 bcc noauxptr ;skip if no carry
CB7A:E6 57 (5) 660 inc auxptr+1 ;don't forget me
CB7C:      661 *
CB7C:      662 * Now look up the first order guess for DIV and MOD. X still has
CB7C:      663 * bytecount DIV 256.
CB7C:      664 *
CB7C:      CB7C 665 noauxptr equ *
CB7C:BD 4C CB (4) 666 lda pdiv7tab,x
CB7F:85 4B (3) 667 sta grp7ctr
CB81:BD 4F CB (4) 668 lda pmod7tab,x
CB84:85 4C (3) 669 sta oddbytes
CB86:      670 *
CB86:      671 * Now add in the mods and divs for each of the five hi order

```

```

CB86:      672 * bits in the lo order bytecount, correcting each time MOD becomes
CB86:      673 * bigger than 6.
CB86:      674 *
CB86:A2 05 (2) 675 ldx #5 ;Do for five bits
CB88:A5 4D (3) 676 lda bytecountl
CB8A:85 59 (3) 677 sta temp ;Store lo order for shifting
CB8C:29 07 (2) 678 and #$00001111 ;Save lo three for later
CB8E:A8 (2) 679 tay
CB8F:      680 *
CB8F:      CB8F 681 divide3 equ *
CB8F:06 59 (5) 682 asl temp ;C <- next from bytecountl
CB91:90 15 CB88(3) 683 bcc divide2 ;If clear, no effect on DIV,MOD
CB93:BD 58 CB (4) 684 lda mod7tab,x ;Get MOD7 for 2^n
CB96:      CB96 685 divide4 equ *
CB96:18 (2) 686 clc
CB97:65 4C (3) 687 adc oddbytes ;Got new MOD value
CB99:C9 07 (2) 688 cmp #7 ;Is it too big?
CB9B:90 02 CB9F(3) 689 blt divide1 ;=> NO leave MOD - 0->C
CB9D:E9 07 (2) 690 sbc #7 ;Bring MOD under 7 - C still set
CB9F:      CB9F 691 divide1 equ *
CB9F:85 4C (3) 692 sta oddbytes
CBAA:BD 52 CB (4) 693 lda div7tab,x ;Get DIV for this 2^n
CBAA:65 4B (3) 694 adc grp7ctr ;Add to DIV along with correction (C)
CBAA:85 4B (3) 695 sta grp7ctr ;Update the DIV
CBAB:      CBAB 696 divide2 equ *
CBAB:CA (2) 697 dex ;One less bit to deal with
CBAB:30 06 CBB1(3) 698 bmi divide5 ;Escape after 6 times through loop
CBAB:D0 E2 CB8F(3) 699 bne divide3 ;Take brnch 1st 5 loops
CBAD:      700 *
CBAD:98 (2) 701 tya ;Get back the last three bits
CBAB:4C 96 CB (3) 702 jmp divide4 ;Sixth pass add in remains
CBB1:      703 *
CBB1:      CBB1 704 divide5 equ *
CBB1:      705 *
CBB1:      706 *

```

05 PC.PACKET

Checksum Prepass

20-OCT-86 06:29 PAGE 27

```

CBB1: 708 *****
CBB1: 709 *
CBB1: 710 * PreCheck Does the checksumming prepass *
CBB1: 711 *
CBB1: 712 * Input: bytecount <- bytes in buffer *
CBB1: 713 * buffer <- pointer to data to send *
CBB1: 714 * auxptr <- extra pointer to speed process *
CBB1: 715 * Output: checksum <- 8 bit XOR of data to be sent *
CBB1: 716 *
CBB1: 717 *****
CBB1: 718 *
CBB1: CBB1 719 PreCheck equ *
CBB1: 720 *
CBB1: 721 * Checksum any Full pages
CBB1: 722 *
CBB1:A5 55 (3) 723 lda buffer+1
CBB3:48 (3) 724 pha ;Preserve buffer pointer
CBB4:A9 00 (2) 725 lda #0
CBB6:A6 4E (3) 726 ldx bytecount
CBB8:F0 16 CBDD(3) 727 beq lastpass ;If no complete pages, skip this
CBB8: CBBA 728 xor2 equ *
CBB8:BC 5E CB (4) 729 ldy auxptrinc,x ;Get number of bytes each ptr
CBB8: CBBD 730 xor1 equ *
CBB8:D1 51 54 (5) 731 eor (buffer),y
CBB8:F1 56 (5) 732 eor (auxptr),y
CBC1:88 (2) 733 dey ;One less
CBC2:D0 F9 CBBD(3) 734 bne xor1
CBC4:51 54 (5) 735 eor (buffer),y
CBC6:51 56 (5) 736 eor (auxptr),y ;Have to deal with 0 case
CBC8: 737 *
CBC8: 738 * Now move the buffer up for next section
CBC8: 739 *
CBC8:E0 01 (2) 740 cpx #1
CBCA:F0 02 CBCE(3) 741 beq xor5 ;If 256 and up bytes, bump x1
CBCC:E6 55 (5) 742 inc buffer+1 ; otherwise x2
CBCE:E6 55 (5) 743 xor5 inc buffer+1
CBDD: 744 *
CBDD: CBDD 745 lastpass equ *
CBDD: 746 *
CBDD: 747 * Do the remaining less than a page with a single pointer
CBDD: 748 *
CBDD:A4 40 (3) 749 ldy bytecount
CBDD:F0 09 CBDD(3) 750 beq xor4
CBDD:51 54 (5) 751 eor (buffer),y ;Compensate for nth byte
CBDD:51 54 (5) 752 xor3 eor (buffer),y
CBDD:88 (2) 753 dey
CBDD:D0 FB CBDD(3) 754 bne xor3
CBDD:51 54 (5) 755 eor (buffer),y ;Last damn (0th) byte
CBDD: 756 *
CBDD: 757 * Store result away. Retrieve old buffer value.
CBDD: 758 *
CBDD: CBDD 759 xor4 equ *
CBDD:85 40 (3) 760 sta checksum
CBDF:68 (4) 761 pla
CBEO:85 55 (3) 762 sta buffer+1
CBE2: 763 *
CBE2: 764 *

```

05 PC.PACKET

Get tophits byte for odds

20-OCT-86 06:29 PAGE 28

```

CBE2: 766 *****
CBE2: 767 *
CBE2: 768 * DetTopBits Get tophits for odd bytes *
CBE2: 769 *
CBE2: 770 * Also sets buffer2 pointer to pointer at groups of *
CBE2: 771 * seven bytes. *
CBE2: 772 *
CBE2: 773 * Input: oddbytes <- # of "odd" bytes *
CBE2: 774 * buffer <- pointer to data *
CBE2: 775 * Output: tbodd <- tophits for odd bytes *
CBE2: 776 * buffer2 <- buffer+oddbytes *
CBE2: 777 *
CBE2: 778 *****
CBE2: 779 *
CBE2: CBE2 780 DetTopBits equ *
CBE2: 781 *
CBE2:A4 4C (3) 782 ldy oddbytes
CBE4:88 (2) 783 dey
CBE5:A9 00 (2) 784 lda #0
CBE7:85 59 (3) 785 sta tbodd
CBE9: 786 *
CBE9:B1 54 (5) 787 gtbob lda (buffer),y
CBE9:0A (2) 788 asl a
CBE9:66 59 (5) 789 ror tbodd
CBE9:88 (2) 790 dey
CBEF:10 F8 CBE9(3) 791 bpl gtbob
CBF1:38 (2) 792 sec
CBF2:66 59 (5) 793 ror tbodd
CBF4: 794 *
CBF4:A5 4C (3) 795 lda oddbytes
CBF6:18 (2) 796 clc
CBF7:65 54 (3) 797 adc buffer
CBF9:85 56 (3) 798 sta buffer2
CBFB:A5 55 (3) 799 lda buffer+1
CBFD:69 00 (2) 800 adc #0
CBFF:85 57 (3) 801 sta buffer2+1
CC01: 802 *
CC01: 803 *

```



```

CC01:      805 *****
CC01:      806 *
CC01:      807 * Sun          Set up next buffer and topbits *
CC01:      808 *
CC01:      809 * Primes the pipe for the group of seven bytes routine *
CC01:      810 * setting the topbits byte and the "next" buffer. *
CC01:      811 * The routine also advances the buffer pointer by 7 to *
CC01:      812 * prepare for the groups of seven transfer. *
CC01:      813 *
CC01:      814 * Input:  buffer2 <- points to groups of 7 data *
CC01:      815 * Output: next1,7 <- first 7 bytes in buffer *
CC01:      816 * topbits <- MSBs of first 7 bytes *
CC01:      817 *
CC01:      818 *****
CC01:      819 *
CC01:      CC01 820 Sun equ *
CC01:      821 *
CC01:      822 * Copy first seven bytes into the pipeline
CC01:      823 *
CC01:A0 06      (2) 824 ldy #6
CC03:38      (2) 825 sun2 sec
CC04:B1 56      (5) 826 lda (buffer2),y
CC06:99 4D 00   (5) 827 sta next,y
CC09:30 01      CC0C(3) 828 bmi sun1
CC0B:18      (2) 829 clc
CC0C:66 41      (5) 830 sun1 ror topbits
CC0E:88      (2) 831 dey
CC0F:10 F2      CC03(3) 832 bpl sun2
CC11:38      (2) 833 sec
CC12:66 41      (5) 834 ror topbits
CC14:      835 *
CC14:      836 * Advance the pointer
CC14:      837 *
CC14:A5 56      (3) 838 lda buffer2
CC16:18      (2) 839 clc
CC17:69 07      (2) 840 adc #7
CC19:85 56      (3) 841 sta buffer2
CC1B:90 02      CC1F(3) 842 bcc sun3
CC1D:E6 57      (5) 843 inc buffer2+1
CC1F:      CC1F 844 sun3 equ *
CC1F:60      (6) 845 rts
CC20:      846 *
CC20:      847 *

```

```

CC20:      849 *
CC20:      850 * X is slot*16, Y is the desired mode
CC20:      851 *
CC20:      852 * Set up the IMM mode register. Extreme care should be taken
CC20:      853 * here. Setting the mode byte with indexed stores causes a
CC20:      854 * false byte to be written a cycle before the real value is
CC20:      855 * written. This false value, if it enables the timer, causes
CC20:      856 * the IMM Rev A to pop the motor on, inhibiting the setting
CC20:      857 * of the mode until the motor times out! We avoid this by
CC20:      858 * setting the mode byte only when it is not what we want, and if
CC20:      859 * it's not we stay here until we see that it is what we want.
CC20:      860 *
CC20:      CC20 861 SetIMMode equ *
CC20:BD 88 C0   (4) 862 lda monclr,x ;Motor must be off
CC23:BD 8D C0   (4) 863 lda l6set,x ;Set up to access mode register
CC26:4C 2D CC   (3) 864 jmp careful ;Don't mess unless we gotta
CC29:98      (2) 865 biz tya
CC2A:9D 8F C0   (5) 866 sta l7set,x ;Try storing the mode value
CC2D:      CC2D 867 careful equ *
CC2D:98      (2) 868 tya ;Get back the target value
CC2E:5D 8E C0   (4) 869 eor l7clr,x ;Compare with observed value
CC31:29 1F      (2) 870 and #51F ;Can only read low 5 bits
CC33:D0 F4      CC29(3) 871 bne biz ;If not right, back to try again
CC35:60      (6) 872 rts
CC36:      873 *
CC36:      874 *
CC36:      CC36 875 WaitIMMOFF equ *
CC36:      876 *
CC36:      877 * Make sure you're in read mode and wait 'til Disk // motor is off
CC36:      878 *
CC36:20 97 CA   (6) 879 jsr SetXN0 ;Set X
CC39:BD 8E C0   (4) 880 lda l7clr,x
CC3C:BD 8D C0   (4) 881 lda l6set,x
CC3F:      CC3F 882 w1wm1 equ *
CC3F:BD 8E C0   (4) 883 lda l7clr,x
CC42:29 20      (2) 884 and #A00100000
CC44:D0 F9      CC3F(3) 885 bne w1wm1
CC46:BD 8C C0   (4) 886 lda l6clr,x
CC49:      887 *
CC49:      888 *Wait an additional 700 usec to allow 12V on Disk // to decay
CC49:      889 *
CC49:5A      (3) 890 phy
CC4A:A0 8C      (2) 891 ldy #140
CC4C:88      (2) 892 w1wm2 dey
CC4D:D0 FD      CC4C(3) 893 bne w1wm2
CC4F:7A      (4) 894 ply
CC50:      895 *
CC50:60      (6) 896 rts
CC51:      897 *
CC51:      898 *
CC51:      899 * This takes grp7ctr and oddbytes and calculates 7*grp7ctr+oddbytes.
CC51:      900 * The results are in Y(hi) and A(lo). This is the number of bytes
CC51:      901 * that were received in the last ReceivePack.
CC51:      902 *
CC51:      CC51 903 Rcvcount equ *
CC51:A5 4B      (3) 904 lda grp7ctr
CC53:A8      (2) 905 tay
CC54:A2 00      (2) 906 ldx #0

```

```

CC56:86 4B (3) 907 stx grp7ctr
CC58:A2 03 (2) 908 ldx #3
CC5A:0A (2) 909 times7 asl a
CC5B:26 4B (5) 910 rol grp7ctr
CC5D:CA (2) 911 dex
CC5E:D0 FA CC5A(3) 912 bne times7
CC60:18 (2) 913 clc
CC61:65 4C (3) 914 adc oddbytes
CC63:90 02 CC67(3) 915 bcc t71
CC65:E6 4B (5) 916 inc grp7ctr
CC67:84 4C (3) 917 t71 sty oddbytes
CC69:38 (2) 918 sec
CC6A:E5 4C (3) 919 sbc oddbytes
CC6C:B0 02 CC70(3) 920 bcs t72
CC6E:C6 4B (5) 921 dec grp7ctr
CC70:A4 4B (3) 922 T72 ldy grp7ctr
CC72:60 (6) 923 rts
CC73: 924 *
CC73: 925 *
CC73: 115 include pc.cread
CC73: CC73 1 SlotDepRd equ *
CC73: CC73 2 start25 equ *
CC73:A0 00 (2) 3 ldy #0
CC75:A5 4B (3) 4 lda grp7ctr
CC77:48 (3) 5 pha ;Save groups of seven counter
CC78:D0 03 CC7D(3) 6 bne start35
CC7A:4C 0A CD (3) 7 jmp done5 ;Go get the checksum
CC7D: 8 *
CC7D: 9 * Okay, get the groups of seven
CC7D: 10 * Start by getting the topbits for this group of seven
CC7D: 11 *
CC7D: CC7D 12 start35 equ *
CC7D:AD EC C0 (4) 13 lda 16clr+TheOff ;Get topbits
CC80:10 FB CC7D(3) 14 bpl start35
CC82:85 59 (3) 15 sta temp ;Just a second
CC84: 16 *
CC84: 17 * Split up the seven bits into two indices for topbit tables
CC84: 18 *
CC84:4A (2) 19 lsr a ;0 1 d1 d2 d3 d4 d5 d6
CC85:4A (2) 20 lsr a ;0 0 1 d1 d2 d3 d4 d5
CC86:4A (2) 21 lsr a ;0 0 0 1 d1 d2 d3 d4
CC87:29 0F (2) 22 and #00001111 ;0 0 0 0 d1 d2 d3 d4
CC89:AA (2) 23 tax ;First index into the tables
CC8A:A5 59 (2) 24 lda temp ;1 d1 d2 d3 d4 d5 d6 d7
CC8C:29 07 (2) 25 and #00000111 ;0 0 0 0 0 d5 d6 d7
CC8E:85 59 (3) 26 sta temp ;Keep for last three bytes
CC90: 27 *
CC90: 28 * Read the 1st byte, reunite its msb, store and checksum it
CC90: 29 *
CC90:AD EC C0 (4) 30 lda 16clr+TheOff
CC93:10 FB CC90(3) 31 bpl *-3 ;Back 1 instruction
CC95:5D 9A CA (4) 32 eor shift1,x ;Recombine the MSB with data
CC98:91 56 (6) 33 sta (buffer2),y ;Store it away
CC9A:45 40 (3) 34 eor checksum ;Add it to the checksum
CC9C:85 40 (3) 35 sta checksum
CC9E:C8 (2) 36 iny
CC9F: 37 *
CC9F: 38 * Now, the second Y turn over occurs at this point in the

```

```

CC9F: 39 * loop. Update the buffer pointer if it occurred.
CC9F: 40 *
CC9F:D0 02 CCA3(3) 41 bne *-4
CCAI:E6 57 (5) 42 inc buffer2+1
CCA3: 43 *
CCA3: 44 * Now the second byte
CCA3: 45 *
CCA3:AD EC C0 (4) 46 lda 16clr+TheOff
CCA6:10 FB CCA3(3) 47 bpl *-3 ;Back 1 instruction
CCAB:5D AA CA (4) 48 eor shift2,x ;Recombine the MSB with data
CCAB:91 56 (6) 49 sta (buffer2),y ;Store it away
CCAD:45 40 (3) 50 eor checksum ;Add it to the checksum
CCAF:85 40 (3) 51 sta checksum
CCB1:C8 (2) 52 iny
CCB2: 53 *
CCB2: 54 * Now the third byte
CCB2: 55 *
CCB2:AD EC C0 (4) 56 lda 16clr+TheOff
CCB5:10 FB CCB2(3) 57 bpl *-3 ;Back 1 instruction
CCB7:5D BA CA (4) 58 eor shift3,x ;Recombine the MSB with data
CCBA:91 56 (6) 59 sta (buffer2),y ;Store it away
CCBC:45 40 (3) 60 eor checksum ;Add it to the checksum
CCBE:85 40 (3) 61 sta checksum
CCD0:C8 (2) 62 iny
CCCI: 63 *
CCCI: 64 * Now the fourth byte
CCCI: 65 *
CCCI:AD EC C0 (4) 66 lda 16clr+TheOff
CCCA:10 FB CCCI(3) 67 bpl *-3 ;Back 1 instruction
CCCB:5D CA CA (4) 68 eor shift4,x ;Recombine the MSB with data
CCCB:91 56 (6) 69 sta (buffer2),y ;Store it away
CCCB:45 40 (3) 70 eor checksum ;Add it to the checksum
CCCD:85 40 (3) 71 sta checksum
CCCF:C8 (2) 72 iny
CCD0: 73 *
CCD0: 74 * The first Y turn over occurs at this point in the loop. Update
CCD0: 75 * the buffer pointer if it occurred.
CCD0: 76 *
CCD0:D0 02 CCD4(3) 77 bne *-4
CCD2:E6 57 (5) 78 inc buffer2+1
CCD4: 79 *
CCD4:A6 59 (3) 80 ldx temp ;Now we need the other index
CCD6: 81 *
CCD6: 82 * Now the fifth byte
CCD6: 83 *
CCD6:AD EC C0 (4) 84 lda 16clr+TheOff
CCD9:10 FB CCD6(3) 85 bpl *-3 ;Back 1 instruction
CCDB:5D AA CA (4) 86 eor shift2,x ;Recombine the MSB with data
CCDE:91 56 (6) 87 sta (buffer2),y ;Store it away
CCDE:45 40 (3) 88 eor checksum ;Add it to the checksum
CEE2:85 40 (3) 89 sta checksum
CEE4:C8 (2) 90 iny
CEES: 91 *
CEES: 92 * Now the sixth byte
CEES: 93 *
CEES:AD EC C0 (4) 94 lda 16clr+TheOff
CEEB:10 FB CEES(3) 95 bpl *-3 ;Back 1 instruction
CEEA:5D BA CA (4) 96 eor shift3,x ;Recombine the MSB with data

```



```

CCED:91 56      (6) 97      sta (buffer2),y ;Store it away
CCF:45 40      (3) 98      eor checksum ;Add it to the checksum
CCF1:85 40     (3) 99      sta checksum
CCF3:C8      (2) 100     iny
CCF4:          101 *
CCF4:          102 * And, finally, the seventh byte
CCF4:          103 *
CCF4:AD EC C0 (4) 104     lda 16clr+TheOff
CCF7:10 FB CCF4(3) 105     bpl *-3 ;Back 1 instruction
CCF9:5D CA CA (4) 106     eor shift4,x ;Recombine the MSB with data
CCFC:91 56     (6) 107     sta (buffer2),y ;Store it away
CCFE:45 40     (3) 108     eor checksum ;Add it to the checksum
CD00:85 40     (3) 109     sta checksum
CD02:C8      (2) 110     iny
CD03:          111 *
CD03:          112 * Now see if this is the last group of seven to receive
CD03:          113 *
CD03:C6 4B     (5) 114     dec grp/ctr
CD05:F0 03 CD0A(3) 115     beq done5 ;Go to get the checksum etc
CD07:4C 7D CC (3) 116     jmp start35 ;Another topbits ...
CD0A:          117 *
CD0A:          118 * Get and reconstruct the checksum
CD0A:          119 *
CD0A:          120 done5 equ *
CD0A:AD EC C0 (4) 121     lda 16clr+TheOff
CD0D:10 FB CD0A(3) 122     bpl *-3
CD0F:85 59     (3) 123     sta temp ;1 c6 1 c4 1 c2 1 c0
CD11:          124 *
CD11:68      (4) 125     pla ;Restore groups of 7 counter
CD12:85 4B     (3) 126     sta grp/ctr
CD14:AD EC C0 (4) 127     lda 16clr+TheOff ;1 c7 1 c5 1 c3 1 c1
CD17:10 FB CD14(3) 128     bpl *-3
CD19:38      (2) 129     sec
CD1A:2A      (2) 130     rol a ;c7 1 c5 1 c3 1 c1 1
CD1B:25 59     (3) 131     and temp ;c7 c6 c5 c4 c3 c2 c1 c0
CD1D:45 40     (3) 132     eor checksum ;When we're done, should be zero
CD1F:          133 *
CD1F:          134 * Get the packet end mark. Is it correct?
CD1F:          135 *
CD1F:AC EC C0 (4) 136     rdha5 ldy 16clr+TheOff ;Preserve A
CD22:10 FB CD1F(3) 137     bpl rdha5
CD24:          138 *
CD24:C0 C8     (2) 139     cpy #packetend
CD26:D0 1C CD44(3) 140     bne npenderr5
CD28:          141 *
CD28:          142 * Didn't have time before to checksum oddbytes. Do it now
CD28:          143 * A still has the partial checksum
CD28:          144 *
CD28:A6 4C     (3) 145     ldx oddbytes
CD2A:F0 08 CD34(3) 146     beq icht15
CD2C:A0 00     (2) 147     ldy #0
CD2E:51 54     (5) 148     icht5 eor (buffer),y
CD30:C8      (2) 149     iny
CD31:CA      (2) 150     dex
CD32:D0 FA CD2E(3) 151     bne icht5
CD34:          152 *
CD34:          153 * Okay, checksum oughta be zero. If not, checksum error.
CD34:          154 *

```

```

CD34:          CD34 155 icht15 equ *
CD34:AA      (2) 156     tax
CD35:D0 11 CD48(3) 157     bne cerror5
CD37:          158 *
CD37:          159 * Wait for /BSY to go low
CD37:          160 *
CD37:          CD37 161 lstbsywait5 equ *
CD37:AD ED C0 (4) 162     lda 16set+TheOff
CD3A:AD EE C0 (4) 163     rdh45 lda 17clr+TheOff
CD3D:30 FB CD3A(3) 164     bmi rdh45
CD3F:          165 *
CD3F:          166 * Got the bytes, now acknowledge their receipt
CD3F:          167 *
CD3F:AD E0 C0 (4) 168     lda reqclr+TheOff ;Lower REQ
CD42:          169 *
CD42:18      (2) 170     clc
CD43:60      (6) 171     rts
CD44:          172 *
CD44:A9 20     (2) 173     npenderr5 lda #nopackend
CD46:D0 02 CD4A(3) 174     bne gerror5
CD48:A9 10     (2) 175     cerror5 lda #csumerr
CD4A:38      (2) 176     gerror5 sec
CD4B:60      (6) 177     rts
CD4C:          178 *
CD4C:          116     include pc.main

```

07 PC.MAIN Protocol Converter / CBus Driver 20-OCT-86 06:29 PAGE 35

```

CD4C:      2 *
CD4C:      3 *
CD4C:      4 Entry equ *
CD4C:90 03 CD51(3) 5 bcc bentry ;If non-boot, skip jump to boot
CD4E:4C 23 C5 (3) 6 jmp bootcode
CD51:      7 *
CD51:      8 * X is still set to slot number.
CD51:      9 *
CD51:      10 bentry equ *
CD51:      11 *
CD51:A9 40 (2) 12 lda #01000000
CD53:1C 78 04 (6) 13 trb ProFlag+5 ;ProFlag is fixed in //c
CD56:      14 *
CD56:      15 atentry equ *
CD56:      16 *
CD56:D8 (2) 17 cld ;Don't want decimal mode!!
CD57:8A (2) 18 txa
CD58:A8 (2) 19 tay ;Really want it in Y... no ROR ABS,Y!
CD59:      20 *
CD59:      21 * If this is a PC call, then get the address of the parm table
CD59:      22 *
CD59:B9 73 04 (4) 23 lda ProFlag,y
CD5C:30 11 CD6F(3) 24 bmi noplay
CD5E:      25 *
CD5E:68 (4) 26 pla ;Get lo order
CD5F:99 F3 05 (5) 27 sta SHTempX,y ;Keep lo parm address-1
CD62:18 (2) 28 clc
CD63:69 03 (2) 29 adc #3
CD65:AA (2) 30 tax ;Lo order new return address
CD66:68 (4) 31 pla ;Get hi order address
CD67:99 73 06 (5) 32 sta SHTempY,y ;Keep hi parm addr-1
CD6A:69 00 (2) 33 adc #0
CD6C:48 (3) 34 pha ;Push back new return address hi
CD6D:8A (2) 35 txa
CD6E:48 (3) 36 pha ;Push new return address lo
CD6F:      37 *
CD6F:      38 noplay equ *
CD6F:      39 *
CD6F:      40 * On the //c, it is important to have the Disk // enable lines
CD6F:      41 * off for as long as possible before using the IWM (phases,
CD6F:      42 * /WRREQ lines). Wait here 'til the Disk // motors are off.
CD6F:      43 *
CD6F:20 36 CC (6) 44 jsr WaitIWMoff ;Must preserve Y!!
CD72:      45 *
CD72:      46 * We can't tolerate ints in most of the code, so disable
CD72:      47 *
CD72:08 (3) 48 php ;Save interrupt status
CD73:78 (2) 49 sei ;No interrupts please
CD74:      50 *
CD74:      51 * Preserve the zero page work area
CD74:      52 *
CD74:A2 18 (2) 53 ldx #ZPSize-1
CD76:B5 40 (4) 54 pzp lda ZeroPage,x
CD78:48 (3) 55 pha
CD79:CA (2) 56 dex
CD7A:10 FA CD76(3) 57 bpl pzp
CD7C:      58 *
CD7C:      59 * Okay, we're safe... now it's all right to store in zero page

```

07 PC.MAIN Protocol Converter / CBus Driver 20-OCT-86 06:29 PAGE 36

```

CD7C:      60 *
CD7C:84 58 (3) 61 sty Slot
CD7E:      62 *
CD7E:      63 *
CD7E:      64 * Now map any ProDOS unit references to our sequential ones.
CD7E:      65 * The method is bizzare and magicians never reveal their secrets.
CD7E:      66 *
CD7E:      67 allset equ *
CD7E:A5 43 (3) 68 lda CMDUnit ;76543210 7&6 specify unit
CD80:2A (2) 69 rol a ;6543210X C<-7
CD81:08 (3) 70 php ;Save drive num
CD82:2A (2) 71 rol a ;543210X7 C<-6
CD83:2A (2) 72 rol a ;43210X76 (6 is grp of 2)
CD84:28 (4) 73 plp ;C<-7
CD85:2A (2) 74 rol a ;3210X767
CD86:29 03 (2) 75 and #00000011 ;ProDOS only installs up to 4
CD88:49 02 (2) 76 eor #00000010 ;000000/67; 6 was /grpof two
CD8A:C0 04 (2) 77 cpy #4 ;If in slot 1,2,or3 reverse grps of two
CD8C:B0 02 CD90(3) 78 bge allset1
CD8E:49 02 (2) 79 eor #00000010
CD90:AA (2) 80 allset1 tax
CD91:E8 (2) 81 inx
CD92:86 43 (3) 82 stx CMDUnit ;You got it
CD94:      83 *
CD94:      84 * Now if this is through the MLI xfase, gotta copy stuff into the
CD94:      85 * send buffer from the parameter list.
CD94:      86 *
CD94:B9 73 04 (4) 87 lda ProFlag,y
CD97:10 03 CD9C(3) 88 bpl darnit
CD99:4C 40 CE (3) 89 jmp skipcopy
CD9C:      90 *
CD9C:      91 * Get the address of the in-line parameter table
CD9C:      92 *
CD9C:      93 darnit equ *
CD9C:B9 F3 05 (4) 94 lda SHTempX,y ;Get back the low part buff addr
CD9F:85 54 (3) 95 sta buffer
CDA1:B9 73 06 (4) 96 lda SHTempY,y ; and the hi part
CDA4:85 55 (3) 97 sta buffer+1
CDA6:      98 *
CDA6:      99 * Now pull out the command code, and the address of the parameters.
CDA6:      100 *
CDA6:A0 01 (2) 101 ldy #1 ;Stacked address is EA-1
CDA8:B1 54 (5) 102 lda (buffer),y
CDA A:85 42 (3) 103 sta cmdcode ;Nice
CDAC:C8 (2) 104 iny
CDAD:B1 54 (5) 105 lda (buffer),y ;Get lo part of parmlist address
CAFA:AA (2) 106 tax ;Save it
CDB0:C8 (2) 107 iny
CDB1:B1 54 (5) 108 lda (buffer),y ;Get hi part
CDB3:85 55 (3) 109 sta buffer+1
CDB5:86 54 (3) 110 stx buffer
CDB7:      111 *
CDB7:      112 * Now buffer points to parmlist
CDB7:      113 * Check command type, and pidgeonhole the parmlist length
CDB7:      114 *
CDB7:A9 01 (2) 115 lda #BadCmd
CDB9:A6 42 (3) 116 ldx cmdcode
CDBB:E0 0A (2) 117 cpx #SA ;Only valid codes are 0-9

```



```

CDB0:90 03 CDC2(3) 118 blt noeh ;=> at least he got that right
CDBF:4C 17 CF (3) 119 ErrorHitch jmp Error ;Gee, maybe we should promote this guy...
CDC2: CDC2 120 noeh equ *
CDC2:A0 00 (2) 121 ldy #0 ;Set for indct compare
CDC4:B1 54 (5) 122 lda (buffer),y ;Get # of parms?
CDC6:85 5A (3) 123 sta Unit
CDC8: 124 *
CDC8: 125 * Now copy the bytes
CDC8: 126 *
CDC8: CDC8 127 okaycnt equ *
CDC8:A0 08 (2) 128 ldy #>cmdlength-1 ;Always copy the maximum
CDCA: CDCA 129 copyloop equ *
CDCA:B1 54 (5) 130 lda (buffer),y ;Pull it out of their hat
CDCC:99 42 00 (5) 131 sta cmdcode,y ;Stuff it into mine
CDF:88 (2) 132 dey
CDD0:D0 F8 CDCA(3) 133 bne copyloop ;Copy 'em all
CDD2: 134 *
CDD2: 135 * Okay. The caller of the PC could be making one of three calls
CDD2: 136 * with a unit number of $00, Control, Init or Status. Check for
CDD2: 137 * these and do what is appropriate.
CDD2: 138 *
CDD2:A5 43 (3) 139 lda CMDUnit
CDD4:D0 6A CE40(3) 140 bne skipcopy ;Never mind
CDD6: 141 *
CDD6: 142 * Check the parameter count for this call to unit#0
CDD6: 143 *
CDD6:A6 42 (3) 144 ldx CMDCode
CDD8:BD 8E CF (4) 145 lda parmctab,x ;Get the length this command
CDD8:29 7F (2) 146 and #$7F ;Force 0 -> MSB
CDD0:A8 (2) 147 tay ;Hang on
CDDE:A9 04 (2) 148 lda #BadPCnt ;Antic bad count
CDE0:C4 5A (3) 149 cpy Unit ;User's pcount is currently here
CDE2:D0 DB CDBF(3) 150 bne ErrorHitch ;What a baby!
CDE4: 151 *
CDE4: 152 * Now service one of the three commands
CDE4: 153 *
CDE4:E0 05 (2) 154 cpx #InitCMD
CDE6:D0 0A CDF2(3) 155 bne notinit ;Not an Init call
CDE8:A9 00 (2) 156 lda #PowerReset ;Just like powerup or reset key(((c)
CDEA:20 98 CF (6) 157 jsr AssignID ;Do a reset cycle
CEDD:A9 00 (2) 158 Aokay lda #0 ;No error allowed
CDF:4C 39 CF (3) 159 jmp sa2
CDF2: 160 *
CDF2:8A (2) 161 notinit txa ;Equiv to 'cmp #StatusCMD'
CDF3:D0 24 CE19(3) 162 bne maybectl
CDF5: 163 *
CDF5:A9 21 (2) 164 lda #BadCtl ;Antic a non zero stat code
CDF7:A6 46 (3) 165 ldx CMDSCCode ;Stat unit#0 can only be code=0
CDF9:D0 C4 CDBF(3) 166 bne ErrorHitch
CDFB: 167 *
CDFB:8A (2) 168 txa ;Equiv to 'lda #0'
CDFC:A6 58 (3) 169 ldx Slot
CDFE:A0 07 (2) 170 ldy #7
CE00:91 44 (6) 171 nini sta (CMDBufferl),y ;Clear some space
CE02:88 (2) 172 dey
CE03:D0 FB CE00(3) 173 bne nini
CE05: 174 *
CE05:BD F9 06 (4) 175 lda NumDevices,x

```

```

CE08:91 44 (6) 176 sta (CMDBufferl),y ;Stick it where they want it
CE0A:C8 (2) 177 iny
CE0B: 178 *
CE0B:AD F9 04 (4) 179 lda $4F9 ;//c Port 1 interrupt status
CE0E: 180 *
CE0E:91 44 (6) 181 sta (CMDBufferl),y ;Store PC interrupt status
CE10: 182 *
CE10:A9 08 (2) 183 lda #8
CE12:88 (2) 184 dey ;A,Y has 0008; # bytes status
CE13:20 F0 CF (6) 185 jsr squirrel
CE16: 186 *
CE16:4C ED CD (3) 187 jmp Aokay ;Skip down (up) with no error
CE19: CE19 188 maybectl equ *
CE19:C9 04 (2) 189 cmp #ControlCMD
CE1B:D0 0B CE28(3) 190 bne BUnit ;Unit #0 was a bad one
CE1D: 191 *
CE1D:A6 46 (3) 192 ldx CMDSCCode ;We allow two control calls for Unit#0
CE1F:F0 0B CE2C(3) 193 beq enabint ;0 means enable interrupts
CE21:CA (2) 194 dex
CE22:F0 14 CE38(3) 195 beq disabint ;1 means disable interrupts
CE24:A9 21 (2) 196 lda #badctl
CE26: CE26 197 ErrorHitch2 equ *
CE26:D0 97 CDBF(3) 198 bne ErrorHitch ;No other codes allowed
CE28: 199 *
CE28: CE28 200 BUnit equ *
CE28:A9 11 (2) 201 lda #badUnit ;Only certain calls can have Unit#0
CE2A:D0 93 CDBF(3) 202 bne ErrorHitch ;Branch always
CE2C: 203 *
CE2C: CE2C 204 enabint equ *
CE2C:A9 C0 (2) 205 lda #$C0
CE2E:8D F9 05 (4) 206 sta $5F9
CE31:A9 0F (2) 207 lda #50F
CE33:0C 9A C0 (6) 208 tsb $C09A
CE36:D0 05 CE3D(3) 209 bne aokayhitch
CE38: 210 *
CE38: CE38 211 disabint equ *
CE38:A9 01 (2) 212 lda #$01
CE3A:1C 9A C0 (6) 213 trb $C09A
CE3D:4C ED CD (3) 214 aokayhitch jmp Aokay
CE40: 215 *
CE40: 216 *
CE40: 217 * Okay, everything's all groovy. ProDOS re-enters here.
CE40: 218 * Check Unit number to be sure there is a corresponding device
CE40: 219 *
CE40: CE40 220 skipcopy equ *
CE40:A9 28 (2) 221 lda #NoDrive ;Anticipate bad unit number
CE42:A4 58 (3) 222 ldy slot
CE44:BE F9 06 (4) 223 ldx NumDevices,y
CE47:E4 43 (3) 224 cpx CMDUnit
CE49:90 DB CE26(3) 225 blt ErrorHitch2 ;Safe- If C clr then Z is clr
CE4B: 226 *
CE4B: 227 * Set buffer and bytecount in anticipation of the inevitable SendPack.
CE4B: 228 *
CE4B:A9 09 (2) 229 lda #>cmdlength
CE4D:85 4D (3) 230 sta bytecount1
CE4F:A9 00 (2) 231 lda #<cmdlength
CE51:85 4E (3) 232 sta bytecountn
CE53:85 55 (3) 233 sta buffer+1

```



```

CE55:A9 42 (2) 234 lda #>cmdcode
CE57:85 54 (3) 235 sta buffer
CE59: 236 *
CE59: 237 * If it's a PC call, omit the next two steps
CE59: 238 *
CE59:A6 58 (3) 239 ldx Slot
CE5B:BD 73 04 (4) 240 lda ProFlag,x ;Is it a call from ProDOS?
CE5E:10 13 CE73(3) 241 bpl notstat ;=> Statcode already set...
CE60: 242 *
CE60: 243 * Need to generate a parameter count for a ProDOS call
CE60: 244 *
CE60:A6 42 (3) 245 ldx CMDCode
CE62:BD 0E CF (4) 246 lda ParmCTab,x
CE65:29 7F (2) 247 and #$7F
CE67:85 5A (3) 248 sta Unit
CE69: 249 *
CE69: 250 * ProDOS always needs the highest blockno byte zeroed
CE69: 251 *
CE69:A9 00 (2) 252 lda #0
CE6B:85 48 (3) 253 sta CMDBlocks
CE6D: 254 *
CE6D: 255 * If this is a ProDOS status call, set stat code to zero
CE6D: 256 *
CE6D:A5 42 (3) 257 lda CMDCode
CE6F:D0 02 CE73(3) 258 bne notstat ;=> Not status so forget it
CE71: 259 *lda #SCDeviceStat ;A is already zero
CE71:85 46 (3) 260 sta CMDSCCode ;Store in command table
CE73: 261 *
CE73: 262 * Okay, finally send over the damn command
CE73: 263 *
CE73: CE73 264 notstat equ *
CE73:A5 5A (3) 265 lda Unit
CE75:A6 43 (3) 266 ldx CmdPCount ;Swap the Parmcount & unit#
CE77:86 5A (3) 267 stx Unit
CE79:85 43 (3) 268 sta CMDPCount ;Now they're correct
CE7B: 269 *
CE7B:A9 80 (2) 270 lda #cmdmark
CE7D:85 5B (3) 271 sta WPacketType
CE7F: 272 *
CE7F:20 87 CA (6) 273 jsr ClrPhases ;Bring all phases off for Quark
CE82: 274 *
CE82:20 E9 CA (6) 275 jsr SendPack
CE85:B0 46 CEC0(3) 276 bcs behitch ;If not okay, skip to bus error
CE87: 277 *
CE87: 278 * Now copy over the buffer address for any data xfer.
CE87: 279 *
CE87:A5 44 (3) 280 lda CMDBuffer
CE89:85 54 (3) 281 sta buffer
CE8B:A5 45 (3) 282 lda CMDBuffer+1
CE8D:85 55 (3) 283 sta buffer+1
CE8F: 284 *
CE8F: 285 * Now for some commands, we have to send over a packet of data, too.
CE8F: 286 * See if this command is one of THOSE.
CE8F: 287 *
CE8F:A6 42 (3) 288 ldx cmdcode
CE91:BD 0E CF (4) 289 lda parmctab,x
CE94:10 3B CED1(3) 290 bpl noxtrasend ;Encoded in top bit
CE96: 291 *

```

```

CE96: 292 * The buffer address and bytecount depend on the call type.
CE96: 293 *
CE96:E0 04 (2) 294 cpx #ControlCmd
CE98:D0 18 CEB2(3) 295 bne NOControl
CE9A: 296 *
CE9A: 297 * In the case of control, bytecount:=(buffer)
CE9A: 298 * and buffer := buffer+2
CE9A: 299 *
CE9A:A0 01 (2) 300 ldy #1
CE9C:B1 54 (5) 301 lda (buffer),y ;Get Hi order bytecount
CE9E:AA (2) 302 tax
CE9F:88 (2) 303 dey
CEA0:B1 54 (5) 304 lda (buffer),y
CEA2:48 (3) 305 pha ;Keep for later
CEA3:18 (2) 306 clc
CEA4:A9 02 (2) 307 lda #2
CEA6:65 54 (3) 308 adc buffer
CEA8:85 54 (3) 309 sta buffer
CEAA:68 (4) 310 pla
CEAB:90 13 CEC0(3) 311 bcc secondsend ;Skip hi ord increment
CEAD:E6 55 (5) 312 inc buffer+1
CEAF:4C C0 CE (3) 313 jmp secondsend ;Skip to store bytecount
CEB2: 314 *
CEB2: CEB2 315 NOControl equ *
CEB2:E0 02 (2) 316 cpx #WriteCMD ;Check for a writeblock
CEB4:D0 06 CEB3(3) 317 bne NOWBlock ;Must be control or write
CEB6: 318 *
CEB6: 319 * In the case of WriteBlock, the length is $12 and the buffer
CEB6: 320 * address is at buffer in the command table
CEB6: 321 *
CEB6:A9 00 (2) 322 lda #0
CEB8:A2 02 (2) 323 ldx #2
CEBA:D0 04 CEC0(3) 324 bne secondsend
CEBC: 325 *
CEBC: 326 * For FileWrite, the buffer address is at CMDbuffer
CEBC: 327 * and the length is at CMDblock.
CEBC: 328 *
CEBC: CEB3 329 NOWBlock equ *
CEBC:A6 47 (3) 330 ldx CMDBlockh
CEBE:A5 46 (3) 331 lda CMDBlockl
CEC0: 332 *
CEC0: CEC0 333 secondsend equ *
CEC0:86 4E (3) 334 stx bytecount
CEC2:85 4D (3) 335 sta bytecount
CEC4: 336 *
CEC4:A9 82 (2) 337 lda #datamark
CEC6:85 5B (3) 338 sta WPacketType ;Identify this as a data packet
CEC8: 339 *
CEC8:20 DA CA (6) 340 jsr SendData
CECB:90 04 CED1(3) 341 bcc noxtrasend
CECD: 342 behitch equ *
CECD:A9 06 (2) 343 lda #BusErr ;This is the bus error hitch
CECF:D0 46 CF17(3) 344 bne Error
CED1: 345 *
CED1: 346 * On ProDOS status call, we've got to point the buffer pointer
CED1: 347 * correctly to zero page... it's the only case special case
CED1: 348 * (on Write, Format and Control no data comes back).
CED1: 349 *

```

```

CED1:      CED1      350 noxtrasend equ *
CED1:A4 58      (3) 351      ldy Slot
CED3:B9 73 04    (4) 352      lda ProFlag,y
CED6:10 0C      CEE4(3) 353      bpl getresults
CED8:A5 42      (3) 354      lda cmdcode
CEDA:D0 08      CEE4(3) 355      bne getresults
CEDC:      356 *
CEDC:A9 45      (2) 357      lda #<CMDBufferh ;Want status in these four
CEDE:A2 00      (2) 358      ldx #<CMDBufferh
CEE0:85 54      (3) 359      sta buffer
CEE2:86 55      (3) 360      stx buffer+1
CEE4:      361 *
CEE4:      362 * Please to be calling ReceivePack
CEE4:      363 *
CEE4:      CEE4      364 getresults equ *
CEE4:20 30 CB    (6) 365      jsr RecPack ;Get status byte (maybe read data too)
CEE7:B0 E4      CEE4(3) 366      bcs behitch
CEE9:      367 *
CEE9:      368 * Figure how many bytes were sent and put that in X,Y temps
CEE9:      369 *
CEE9:20 51 CC    (6) 370      jsr Rcvcount ;Do the times 7...
CEEC:20 F0 CF    (6) 371      jsr squirrel ;Store away count in SHTEMPs
CEEF:      372 *
CEEF:      373 * For the ProDOS status call, we've got to look at the status byte
CEEF:      374 * returned and return a DIP error if appropriate. Also overwrite
CEEF:      375 * the X,Y temps with # blocks if this is a ProDOS Stat call.
CEEF:      376 *
CEEF:A5 42      (3) 377      lda CMDCode ;Is it a ProDOS status call
CEFI:D0 22      CF15(3) 378      bne noerror
CEF3:A6 58      (3) 379      ldx Slot
CEF5:BD 73 04    (4) 380      lda ProFlag,x
CEF8:10 1B      CF15(3) 381      bpl noerror
CEFA:      382 *
CEFA:A5 46      (3) 383      lda CMDBlockl ;This'll get loaded into the XY regs later
CEFC:9D F3 05    (5) 384      sta SHTempX,x
CEFF:A5 47      (3) 385      lda CMDBlockh
CF01:9D 73 06    (5) 386      sta SHTempY,x
CF04:      387 *
CF04:A5 45      (3) 388      lda CMDBufferh ;Check status byte
CF06:4A      (2) 389      lsr a
CF07:4A      (2) 390      lsr a
CF08:4A      (2) 391      lsr a
CF09:90 04      CF0F(3) 392      bcc ChkOffLn ;no error, go check off line
CF0B:A9 2B      (2) 393      lda #WriteProt ;else set WPROT error
CF0D:80 08      CF17(3) 394      bra error
CF0F:      CF0F      395 ChkOffLn equ *
CF0F:4A      (2) 396      lsr a
CF10:4A      (2) 397      lsr a
CF11:A9 2F      (2) 398      lda #Offline ;Assume error
CF13:90 02      CF17(3) 399      bcc error
CF15:      400 *
CF15:      401 * Now it's time to think about returning to the caller
CF15:      402 * Remember that ProDOS doesn't want to know about soft errors,
CF15:      403 * only fatal ones. If this is a ProDOS call, and the soft error
CF15:      404 * bit in the statbyte is set, there IS NO error (statbyte is
CF15:      405 * cleared). Also, ProDOS wants only I/O, Write Protect, No Device,
CF15:      406 * Offline. If any other hard error comes from the device
CF15:      407 * on a ProDOS call, map it to an I/O Error. (Gross me out.)

```

```

CF15:      408 *
CF15:      CF15      409 noerror equ *
CF15:A5 4D      (3) 410      lda statbyte
CF17:      CF17      411 Error equ *
CF17:A4 58      (3) 412      ldy Slot ;Need access to screenholes
CF19:99 F3 04    (5) 413      sta Retry,Y ;Keep unadulterated error in shole
CF1C:AA      (2) 414      tax ;Set the Z flag
CF1D:F0 1A      CF39(3) 415      beq sa2 ;Special case the zero
CF1F:      416 *
CF1F:BE 73 04    (4) 417      ldx ProFlag,y ;Set N to ProDOS call or not
CF22:10 15      CF39(3) 418      bpl sa2 ;If PC call, no mapping occurs
CF24:      419 *
CF24:A2 00      (2) 420      ldx #0 ;Assume a soft error
CF26:C9 40      (2) 421      cmp #01000000 ;Soft error check
CF28:B0 0E      CF38(3) 422      bge storeaway ;If $40 or bigger, map to zero
CF2A:      423 *
CF2A:A2 27      (2) 424      ldx #IOError ;Now anticipate ProDOS I/O error
CF2C:C9 2B      (2) 425      cmp #WriteProt
CF2E:F0 09      CF39(3) 426      beq sa2 ;OK to return Write Protect
CF30:C9 28      (2) 427      cmp #NoDrive
CF32:F0 05      CF39(3) 428      beq sa2 ;OK to return Drive disconnected
CF34:C9 2F      (2) 429      cmp #Offline
CF36:F0 01      CF39(3) 430      beq SA2
CF38:      431 *
CF38:      CF38      432 storeaway equ *
CF38:8A      (2) 433      txa ;Use the default value
CF39:      CF39      434 sa2 equ *
CF39:A4 58      (3) 435      ldy Slot
CF3B:99 73 05    (5) 436      sta SHTempY ;Keep in screenhole
CF3E:      437 *
CF3E:      438 * If this is the //c version, we need to reset the IMM to its
CF3E:      439 * former disk // state. This is done by setting the mode register
CF3E:      440 * to a little known (and less documented) mode which speeds up the
CF3E:      441 * internal motor timeout. When the motor enable has timed out, the
CF3E:      442 * mode can be set back to zero. This method is necessary because
CF3E:      443 * if the timer is enabled within the timeout period, the motor on a
CF3E:      444 * Rev A IMM pops on for the full timeout period (since mode changes
CF3E:      445 * are disabled when the motor is on. It's bizzarre. Blame Mac.
CF3E:AD E8 C0    (4) 446      lda monclr+$60 ;Motor off
CF41:2C ED C0    (4) 447      bit 16set+$60 ;Into mode reg access mode
CF44:A9 2B      (2) 448      lda #S2B ;This is the magic "speed up" value
CF46:BD EF C0    (4) 449      sta 17set+$60 ;Throw into mode register
CF49:EA      (2) 450      nop ;You're supposed to wait a while
CF4A:EA      (2) 451      nop
CF4B:EA      (2) 452      nop
CF4C:EA      (2) 453      nop
CF4D:      CF4D      454 waitoff equ *
CF4D:AD EE C0    (4) 455      lda 17clr+$60 ;Wait 'til motor off
CF50:29 20      (2) 456      and #S20
CF52:D0 F9      CF4D(3) 457      bne waitoff
CF54:A0 00      (2) 458      ldy #0 ;Now set the reg back to $00
CF56:A2 60      (2) 459      ldx #S60 ;IMM's in slot 6
CF58:20 20 CC    (6) 460      jsr SetIMMode
CF5B:AD EC C0    (4) 461      lda 16clr+$60
CF5E:AD E2 C0    (4) 462      lda 16calclr+$60
CF61:AD E6 C0    (4) 463      lda 16strbclr+$60
CF64:A4 58      (3) 464      ldy Slot ;Need Slot in Y
CF66:      465 *

```



```

CF66:      466 * Now, restore our zero page area.
CF66:      467 *
CF66:A2 00 (2) 468 ldx #0
CF68:68 (4) 469 rzp pla zeropage,x
CF69:95 40 (4) 470 sta zeropage,x
CF68:E8 (2) 471 inx
CF6C:E0 1C (2) 472 cpx #ZPSize
CF6E:90 F8 CF68(3) 473 blt rzp
CF70:      474 *
CF70:      475 * We're into the stretch! Restore interrupt mask, load X, Y,
CF70:      476 * and A and set the carry if the error byte is non-zero.
CF70:      477 *
CF70:28 (4) 478 plp ;Restore interrupt flag
CF71:B9 F3 05 (4) 479 lda SHTempx,y ;Get X value
CF74:AA (2) 480 tax
CF75:B9 73 05 (4) 481 lda SHTemp,y ;Grab the error result code
CF78:48 (3) 482 pha
CF79:B9 73 06 (4) 483 lda SHTemp,y ;Pull out the Y value
CF7C:A8 (2) 484 tay ;No more access to screenholes
CF7D:18 (2) 485 clc ;Anticipate zero result code
CF7E:68 (4) 486 pla ;Pull back result code
CF7F:F0 01 CF82(3) 487 beq finalskip ;Return with carry clear
CF81:38 (2) 488 sec ;Some type of error
CF82:      CF82 489 finalskip equ *
CF82:      490 *
CF82:08 (3) 491 php ;Save carry and Z flag
CF83:2C 78 04 (4) 492 bit ProFlag+5 ;Ick - ProFlag is fixed in //c
CF86:70 04 CF8C(3) 493 bvs ickl ;If bit 6=1, then return to alt ROM
CF88:28 (4) 494 plp ;Vclr so return across ROM bank bdy
CF89:4C 84 C7 (3) 495 jmp SWRTS2
CF8C:      CF8C 496 ickl equ *
CF8C:28 (4) 497 plp
CF8D:60 (6) 498 rts ;Flags set correctly again
CF8E:      499 *
CF8E:      500 *
CF8E:      CF8E 501 parmctab equ *
CF8E:03 502 dfb $00000011 ;Status: 3 parms/no data send
CF8F:03 503 dfb $00000011 ;Read: 3 parms/no data send
CF90:83 504 dfb $10000011 ;Write: 3 parms/data send
CF91:01 505 dfb $00000001 ;Format: 1 parm /no data send
CF92:83 506 dfb $10000011 ;Control: 3 parms/data send
CF93:01 507 dfb $00000001 ;Init: 1 parm /no data send
CF94:01 508 dfb $00000001 ;Open: 1 parm /no data send
CF95:01 509 dfb $00000001 ;Close: 1 parm /no data send
CF96:03 510 dfb $00000011 ;CharRead: 3 parms/data send
CF97:83 511 dfb $10000011 ;CharWrite: 3 parms/data send
CF98:      512 *
CF98:      513 *

```

```

CF98:      515 *
CF98:      CF98 516 AssignID equ *
CF98:48 (3) 517 pha ;Save the init code
CF99:20 5D CA (6) 518 jsr resetchain ;Reset all of those things
CF9C:68 (4) 519 pla
CF9D:AA (2) 520 tax ;Save InitCode
CF9E:      521 *
CF9E:      522 * Save the command code, unit, and init code
CF9E:      523 * 'cause we'll trample 'em.
CF9E:      524 *
CF9E:A5 42 (3) 525 lda CMDCode
CFA0:48 (3) 526 pha
CFA1:A5 43 (3) 527 lda CMDPCount
CFA3:48 (3) 528 pha
CFA4:A5 46 (3) 529 lda CMDSCCode
CFA6:48 (3) 530 pha
CFA7:86 46 (3) 531 stx CMDSCCode ;Store away the type of INIT
CFA9:      532 *
CFA9:      533 * Set up to send DefID command packets
CFA9:      534 *
CFA9:A9 05 (2) 535 lda #InitCmd
CFAB:85 42 (3) 536 sta CMDCode
CFAD:A9 00 (2) 537 lda #0
CFAF:85 5A (3) 538 sta Unit
CFB1:A9 02 (2) 539 lda #2 ;# parms in Init call
CFB3:85 43 (3) 540 sta CMDPCount
CFB5:      541 *
CFB5:      542 * Point the buffer pointer
CFB5:      543 *
CFB5:A9 42 (2) 544 lda #>CMDCode
CFB7:85 54 (3) 545 sta buffer
CFB9:A9 00 (2) 546 lda #<CMDCode
CFBB:85 55 (3) 547 sta buffer+1
CFBD:A9 80 (2) 548 lda #cmdmark
CFBF:85 5B (3) 549 sta WPacketType
CFC1:      550 *
CFC1:20 87 CA (6) 551 jsr ClrPhases ;Make sure phases are off for Quark
CFC4:      552 *
CFC4:      553 * Send an ID for the next device in the chain
CFC4:      554 *
CFC4:      CFC4 555 mordevices equ *
CFC4:E6 5A (5) 556 inc Unit
CFC6:A9 09 (2) 557 lda #>cmdlength
CFC8:85 4D (3) 558 sta bytecount1 ;ReceivePack scrambles count
CFC9:A9 00 (2) 559 lda #<cmdlength
CFCC:85 4E (3) 560 sta bytecountn
CFCE:      561 *
CFCE:20 83 C8 (6) 562 jsr SendOnePack ;Send the command
CFD1:90 05 CFD8(3) 563 bcc mdev2 ;If okay, skip to get response
CFD3:      564 *
CFD3:C6 5A (5) 565 dec Unit
CFD5:4C DF CF (3) 566 jmp mdev1
CFD8:      567 *
CFD8:20 E3 C9 (6) 568 mdev2 jsr ReceivePack ;Get the response
CFDB:A5 4D (3) 569 lda statbyte
CFDD:F0 E5 CFC4(3) 570 beq mordevices
CFDF:      571 *
CFDF:      572 * Okay, we done last device. Squirrel away the number of devices.

```



```

CFDF:          573 *
CFDF:A5 5A     (3) 574 mdevl lda Unit
CFE1:A4 58     (3) 575 ldy slot
CFE3:99 F9 06 (5) 576 sta NumDevices,y ;Devices out there
CFE6:          577 *
CFE6:          578 * Recover the scrambled ProDOS parms
CFE6:          579 *
CFE6:68        (4) 580 pla
CFE7:85 46     (3) 581 sta CMDSCode
CFE9:68        (4) 582 pla
CFE9:85 43     (3) 583 sta CMDPCount
CFEC:68        (4) 584 pla
CFED:85 42     (3) 585 sta CMDCode
CFEF:          586 *
CFEF:60        (6) 587 rts
CFF0:          588 *
CFF0:          589 *
CFF0:          CFF0 590 squirrel equ *
CFF0:A6 58     (3) 591 ldx Slot
CFF2:9D F3 05 (5) 592 sta SHTempX,x
CFF5:98        (2) 593 tya
CFF6:9D 73 06 (5) 594 sta SHTempY,x
CFF9:60        (6) 595 rts
CFFA:          596 *
CFFA:          597 *

```

| | | | |
|------------------|-------------------|------------------|-------------------|
| C908 ACHE1 | ?CD7E ALLSET | CD90 ALLSET1 | ?CB01 ALTSENOPILE |
| CDED AOKAY | CE3D AOKAYHITCH | CF98 ASSIGNID | ?CD56 ATENTRY |
| ?FABA AUTOSCAN | 56 AUXPTR | CB5E AUXPTRINC | ? 4E AUXTYPE |
| ? 2D BADBLOCK | 01 BADCMD | ? 22 BADCTLPAARM | 21 BADCTL |
| 04 BADPCNT | 11 BADUNIT | ?E000 BASIC | C52F BCI |
| CECD BEHITCH | CD51 BENTRY | CC29 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 | CE28 BUNIT |
| 06 BUSER | ? 40 BUSBOG | ? 08 BYTECMP | 4D BYTECOUNT |
| 4E BYTECOUNTH | 4D BYTECOUNTL | ?C500 C500ORG | C082 CAICLR |
| C083 CAISET | C084 CAZCLR | C085 CAZSET | CC2D CAREFUL |
| CF0F CHNOFFLN | ? 24 CH | C8A2 CHAINUNBSY | 40 CHECKSUM |
| CFFF CLEARIORCMS | CA87 CLRPHASES | 47 CMBLOCKH | ? 46 CMBLOCK |
| 46 CMBLOCKL | 48 CMBLOCKS | 45 CMBUFFERH | 44 CMBUFFERL |
| 44 CMBUFFER | 42 CMDCODE | 09 CMDLENGTH | ?C58A CMDLIST |
| 80 CMDMARK | 43 CMDPCOUNT | 46 CMDSCODE | ? 49 CMDSPARE1 |
| ? 4A CMDSPARE2 | 43 CMDUNIT | C55D COMA | 80 COMRESET |
| 04 CONTROLCMD | CDCA COPYLOOP | ?FDED COUT | CD48 CSERRORS |
| 10 CSUMERR | ? 25 CV | CD9C DARNIT | 82 DATAMARK |
| C996 DATONE | C9C0 DBERROR | ?CBE2 DETTOPBITS | ? 50 DEVICEID |
| CE38 DISABINT | CB52 DIV7TAB | CB9F DIVIDE1 | CBAB DIVIDE2 |
| CB8F DIVIDE3 | CB96 DIVIDE4 | CB81 DIVIDE5 | ?CB61 DIVIDE7 |
| CD0A DONES | CE2C ENABINT | ?C08A ENABLE1 | C08B ENABLE2 |
| CA7D ENABLECHAIN | CD4C ENTRY | CDBF ERRORHITCH | CF17 ERROR |
| CE26 ERRORHITCH2 | CF82 FINALSkip | ? 03 FORMATCMD | CEE4 GETRESULTS |
| CA44 G0B1 | ?C9E3 GRABSTATUS | 4B GRP7CTR | CD4A GSERRORS |
| CBE9 GTR0B | ? 51 HOSTID | CD34 ICBT15 | CD2E ICBT5 |
| CF8C ICK1 | 05 INITCMD | 27 IOERROR | C080 IMM |
| 07 INMMODE | C08C L6CLR | C08D L6SET | C08E L7CLR |
| C08F L7SET | ? 68 LASTONE | CBDD LASTPASS | 00 LOCO |
| ? 01 LOC1 | ?CD37 LSTBSYMAITS | C086 LSTRBCLR | C087 LSTRBSET |
| C9E0 MARKERR | CE19 MAYBECTRL | CFDF MDEV1 | CFD8 MDEV2 |
| C500 MLIENTRY | CB58 MOD7TAB | C088 MONCLR | C089 MONSET |
| C554 MORCHRS | CFC4 MORDEVICES | 07F8 MSL0T | 4D NEXT1 |
| 4E NEXT2 | 4F NEXT3 | 50 NEXT4 | 51 NEXT5 |
| 52 NEXT6 | 53 NEXT7 | 4D NEXT | CE00 NIN1 |
| 01 NOANSWER | CB7C NOAUXPTR | CEB2 NOCONTROL | 28 NODRIVE |
| CDC2 NOH | CF15 NOERROR | ? 1F NOINT | ? 02 NOMARK |
| 20 NOPACKEND | CD6F NOPLAY | CD62 NOTINIT | CE73 NOTSTAT |
| CEBC NOWBLOCK | CE01 NOXTRASEND | CD44 NPENDERR5 | 06F9 NUMDEVICES |
| 4C ODDBYTES | 2F OFFLINE | ?CDC8 OKAYCNT | CA79 ONEMS1 |
| CA77 ONEMS | C3 PACKETBEG | C8 PACKETEND | CF8E PARMCTAB |
| C9BB PATCH1 | ? A5 PBBVALUE | ? FF PBCVALUE | 00 PCID2 |
| BF PDIDBYTE | CB4C PDIV7TAB | CB4F PMOD7TAB | ? 52 POINTER |
| 00 POWERRESET | C9D3 PREAMBLE | ?CB81 PRECHECK | C50A PRODOSENTRY |
| 0473 PROFILAG | CD76 P2P | 0BB8 RCI | 05 RC2 |
| C583 RC0DE | 4B RCVBUF | CC51 RCVCOUNT | C9F5 RDH1 |
| C9FF RDH2 | CA0D RDH3 | CD3A RDH45 | ?CA0B RDH5 |
| CD1F RDH5 | 01 READCMD | C9E3 RECEIVEPACK | CB30 RECPACK |
| C080 REQCLR | C081 REQSET | C576 RESET | CASD RESETCHAIN |
| 04F3 RETRY | 0573 RETRY2 | ? 4F RPACKETTYPE | CB37 RPK1 |
| CB4B RPOUT | C578 RST1 | CF68 R2P | CF39 SA2 |
| CB73 SAP1 | ? 00 SCDEVICESTAT | ? 01 SCGETDCB | ? 03 SCGETDEVINFO |
| 0473 SCHOLES | C99A SCM1 | ? 02 SCRETNLSTAT | C9CC SD10 |
| C9AF SD7 | C9C6 SD9 | CAE8 SDOUBT | CEC0 SECONDSEND |
| CA4E SEND80 | CAS0 SENDBYTE | CADA SENDDATA | C883 SENDONEPACK |
| CAE9 SENDPACK | CAFD SENDPILE | CC20 SETIWMODE | ?FE89 SETKBD |

07 SYMBOL TABLE SORTED BY SYMBOL 20-OCT-86 06:29 PAGE 47

| | | | |
|----------------|----------------|----------------|-----------------|
| ?FE93 SETVID | CA97 SETXN0 | CA9A SHIF1 | CAAA SHIF2 |
| CABA SHIF3 | CACA SHIF4 | 0573 SHTEMP1 | 05F3 SHTEMPX |
| 0673 SHTEMPY | C924 SKIP1 | C926 SKIP2 | C960 SKIP3 |
| C962 SKIP4 | CE40 SKIPCOPY | CC73 SLOTPEDRD | 58 SLOT |
| C8E5 SOB1 | C8F6 SOB2 | C8FD SOB3 | 40 SOFT |
| ? 67 SOFTERROR | CB0A SPIEL1 | CB2F SPILOUT | CFFO SQUIRREL |
| C8AC SSB | C8AF SSD | ?0100 STACK | CA31 START0 |
| CA39 START1 | CA4B START2 | ?CC73 START25 | CC7D START35 |
| C900 START | 4D STATBYTE | ? 81 STATMARK | 1E STATMTO |
| ? 00 STATUSCMD | CF38 STOREAWAY | CC03 SUN2 | ?CC01 SUN |
| CC0C SUN1 | CC1F SUN3 | 0778 SVBCH | 06F8 SVBCL |
| ? 10 SVMASK1 | C797 SWPROTO | C784 SWRTS2 | ?C9D4 SYNCTAB |
| CC67 T71 | CC70 T72 | 59 TBODD | 59 TEMP |
| 60 THEOFF | CC5A TIMES7 | 41 TOPBITS | C896 UBSY1 |
| 5A UNIT | ?0101 VERSION | ?FC22 VTAB | CC36 WAITIMMOFF |
| CF4D WAITOFF | ? 04 WASRESET | ?C9DF WASTE12 | C9DE WASTE14 |
| ?C9DD WASTE16 | ?C9DC WASTE18 | ?C9D9 WASTE32 | CC3F WINM1 |
| CC4C WINM2 | 5B WPACKETTYPE | 02 WRITEMCD | CB61 WRITEPREP |
| 2B WRITEPROT | CBBD XOR1 | ?CBBA XOR2 | CBDE XOR3 |
| CBDD XOR4 | CBCE XOR5 | CA70 YMSWAIT | 40 ZEROPAGE |
| 1C ZPSIZE | | | |

07 SYMBOL TABLE SORTED BY ADDRESS 20-OCT-86 06:29 PAGE 48

| | | | |
|-------------------|-------------------|-------------------|-------------------|
| ? 00 STATUSCMD | ? 00 SCDEVICESTAT | 00 PCID2 | 00 LOCO |
| 00 POWERRESET | ? 01 LOC1 | 01 BADCMD | ? 01 SCGETDCB |
| 01 READCMD | 01 NOANSWER | ? 02 NOMARK | 02 WRITEMCD |
| ? 02 SECRETNLSTAT | ? 03 FORMATCMD | ? 03 SCGETDEVINFO | ? 04 WASRESET |
| 04 CONTROLCMD | 04 BADPCNT | 05 RC2 | 05 INITCMD |
| 06 BUSERR | 07 IMMMODE | ? 08 BYTECMP | 09 CMDLENGTH |
| 0A BSYT02 | 10 CSUMERR | ? 10 SVMASK1 | 0011 BMSGLEN |
| 11 BADUNIT | 1C ZPSIZE | 1E STATMTO | ? 1F NOINT |
| 20 NOPACKEND | 21 BADCTL | ? 22 BADCTLPRM | ? 24 CR |
| ? 25 CV | 27 IOERROR | 28 NODRIVE | 2B WRITEPROT |
| ? 2D BADBLOCK | 2F OFFLINE | 32 BSYT01 | 40 ZEROPAGE |
| ? 40 BUSHOG | 40 CHECKSUM | 40 SOFT | 41 TOPBITS |
| 42 CMDCODE | 43 CMDPCOUNT | 43 CMDUNIT | 44 CMDBUFFER |
| 44 CMDBUFFERL | 45 CMDBUFFERH | 46 CMDSCODE | ? 46 CMBLOCK |
| 46 CMBLOCKL | 47 CMBLOCKH | 48 CMBLOCKS | ? 49 CMBSPARE1 |
| ? 4A CMBSPARE2 | 4B RCVBUFF | 4B GRP7CTR | 4C ODDBYTES |
| 4D NEXT1 | 4D BYTECOUNTL | 4D NEXT | 4D STATBYTE |
| 4D BYTECOUNT | 4E NEXT2 | 4E BYTECOUNTH | ? 4E AUXTYPE |
| ? 4F RPACKETTYPE | 4F NEXT3 | 50 NEXT4 | ? 50 DEVICEID |
| ? 51 BOSTID | 51 NEXT5 | 52 NEXT6 | ? 52 POINTER |
| 53 NEXT7 | 54 BUFFER | 56 AUXPTR | 56 BUFFER2 |
| 58 SLOT | 59 TBODD | 59 TEMP | 5A UNIT |
| 5B WPACKETTYPE | 60 THEOFF | ? 67 SOFTERROR | ? 68 LASTONE |
| 80 COMMRSET | 80 CMDMARK | ? 81 STATMARK | 82 DATAMARK |
| ? A5 PBBVALUE | BF PDIDBYTE | C3 PACKETHEG | C8 PACKETEND |
| ? FF PBCVALUE | ?0100 STACK | ?0101 VERSION | 0473 PROFLAG |
| 0473 SCHOLES | 04F3 RETRY | 0573 RETRY2 | 0573 SHTEMP1 |
| 05F3 SHTEMPX | 0673 SHTEMPY | 06F8 SVBCL | 06F9 NUMDEVICES |
| 0778 SVBCH | 07DB BOOTSCRN | 07F8 MSL0T | 0888 RC1 |
| C080 REQCLR | C080 IMM | C081 REQSET | C082 CAICLR |
| C083 CA1SET | C084 CAZCLR | C085 CAZSET | C086 LSTRBCLR |
| C087 LSTRBSET | C088 MONCLR | C089 MONSET | ?C08A ENABLE1 |
| C08B ENABLE2 | C08C L6CLR | C08D L6SET | C08E L7CLR |
| C08F L7SET | ?C500 CS00ORG | C50A PRODOSENTRY | C50D MLIENTRY |
| ?C514 BOOTCASE5 | C521 BOOTC | C523 BOOTCODE | C52F BC1 |
| C552 BOOTFAIL | C554 MORCHRS | C55D COMA | C55F BOOTMSG |
| C570 BOOTTAB | C576 RESET | C578 RST1 | C583 RCODE |
| ?C58A CMDLIST | C784 SWRTS2 | C797 SWPROTO | C883 SENDONEPACK |
| C896 UBSY1 | C8A2 CHAINUNBSY | C8AC SSB | C8AF SSD |
| C8E5 SOB1 | C8F6 SOB2 | C8FD SOB3 | C900 START |
| C90B ACHE1 | C924 SKIP1 | C926 SKIP2 | C960 SKIP3 |
| C962 SKIP4 | C996 DATDONE | C99A SCMI | C9AF SD7 |
| C98B PATCH1 | C9C0 DBERROR | C9C6 SD9 | C9CC SD10 |
| C903 PREAMBLE | ?C9D4 SYNCTAB | ?C9D9 WASTE32 | ?C9DC WASTE18 |
| ?C9DD WASTE16 | C9DE WASTE14 | ?C9DF WASTE12 | C9E0 MARKERR |
| C9E3 RECEIVEPACK | ?C9E3 GRABSTATUS | C9F5 RDH1 | C9FF RDH2 |
| ?CA0B RDB5 | CA0D RDH3 | CA31 START0 | CA39 START1 |
| CA44 G0B1 | CA4B START2 | CA4E SEND80 | CA50 SENDBYTE |
| CA5D RESETCHAIN | CA70 YMSWAIT | CA77 ONEMS | CA79 ONEMS1 |
| CA7D ENABLECHAIN | CA87 CLRPHASES | CA97 SETXN0 | CA9A SHIF1 |
| CAAA SHIF2 | CABA SHIF3 | CACA SHIF4 | CADA SENDDATA |
| CAE8 SDOUBT | CAE9 SENDPACK | CAFD SENDPILE | ?CB01 ALTSENDPILE |
| CB0A SPIEL1 | CB2F SPILOUT | CB30 REPACK | CB37 RPK1 |
| CB4B RPOUT | CB4C PDIV7TAB | CB4F PMOD7TAB | CB52 DIV7TAB |
| CB58 MOD7TAB | CB5E AUXPTRINC | CB61 WRITEPREP | ?CB61 DIVIDE7 |
| CB73 SAP1 | CB7C NOAUXPTR | CB8F DIVIDE3 | CB96 DIVIDE4 |
| CB9F DIVIDE1 | CBAB DIVIDE2 | ?CB81 PRECHECK | CB81 DIVIDE5 |
| ?CBBA XOR2 | CBBD XOR1 | CBCE XOR5 | CBDD LASTPASS |

| | | | |
|----------------|-------------------|------------------|-----------------|
| CB06 XOR3 | CB0D XOR4 | ?CBE2 DETTOPBITS | CBE9 GTBOB |
| ?CC01 SUN | CC03 SUN2 | CC0C SUN1 | CC1F SUN3 |
| CC20 SETIMMODE | CC29 BIZ | CC2D CAREFUL | CC36 WAITIMMOFF |
| CC3F WIMM1 | CC4C WIMM2 | CC51 RVCOUNT | CC5A TIMES7 |
| CC67 T71 | CC70 T72 | CC73 SLOTDEPRD | ?CC73 START25 |
| CC7D START35 | CD0A DONE5 | CD1F RDHA5 | CD2E ICBT5 |
| CD34 ICBT15 | ?CD37 LSTBSYWAITS | CD3A RDH45 | CD44 NPENDERR5 |
| CD48 CSERROR5 | CD4A GSERROR5 | CD4C ENTRY | CD51 BENTRY |
| ?CD56 AENTRY | CD6F NOPLAY | CD76 PZP | ?CD7E ALLSET |
| CD90 ALLSET1 | CD9C DARNIT | CDBF ERRORHITCH | CDC2 NOEH |
| ?CDC8 OKAYCNT | CDCA COPYLOOP | CDED AOKAY | CDF2 NOTINIT |
| CE00 NIN1 | CE19 MAYBECTRL | CE26 ERRORHITCH2 | CE28 BUNIT |
| CE2C ENABINT | CE38 DISABINT | CE3D AOKAYHITCH | CE40 SKIPCOPY |
| CE73 NOTSTAT | CEB2 NOCONTROL | CEBC NOWBLOCK | CEC0 SECONDSEND |
| CEED BEHITCH | CEB1 NOXTRASEND | CEE4 GETRESULTS | CF0F CHKOFFLN |
| CF15 NOERROR | CF17 ERROR | CF38 STOREAWAY | CF39 SA2 |
| CF40 WAITOFF | CF68 RZP | CF82 FINALSkip | CF8C ICK1 |
| CF8E PARMCTAB | CF98 ASSIGNID | CFC4 MORDEVICES | CFD8 MDEV2 |
| CFDF MDEV1 | CFF0 SQUIRREL | CFFF CLEARIOROMS | ?E000 BASIC |
| ?FABA AUTOSCAN | ?FC22 VTAB | ?FDED COUT | ?FE89 SETKBD |

** SUCCESSFUL ASSEMBLY := NO ERRORS
 ** ASSEMBLER CREATED ON 15-JAN-84 21:28
 ** TOTAL LINES ASSEMBLED 2157
 ** FREE SPACE PAGE COUNT 70