```
1   ************************************************************
2   *                                                          *
3   *                  P a s s i v e B o o t                   *
4   *                     for NadaNet 3.x                      *
5   *                                                          *
6   *              Michael J. Mahon - June 23, 2004            *
7   *                   Adapted Apr 15, 2008                   *
8   *                   Revised Apr 30, 2010                   *
9   *                                                          *
10  *    Copyright (c) 1996, 2003, 2004, 2005, 2008, 2010      *
11  *                                                          *
12  *   PassiveBoot is ROM boot code for a diskless Enhanced   *
13  *   Apple //e which replaces the self-test code.  It       *
14  *   causes the //e to boot from a "master-capable"         *
15  *   machine on the network that sends the boot code.       *
16  *                                                          *
17  *   This major modification to the boot protocol creates   *
18  *   a minimal boot code which can fit into the smaller     *
19  *   2-page self-test area of the Enhanced //e ROM.         *
20  *                                                          *
21  *   This version only waits for a BOOT transaction to      *
22  *   appear on the net, then it loads the boot image        *
23  *   sent and passes it control.                            *
24  *                                                          *
25  *   This allows all but RCVPKT and RCVLONG, plus the       *
26  *   RESET and boot logic itself to be eliminated.          *
27  *                                                          *
28  ************************************************************
29  *                                                          *
30  *                     Change History                       *
31  *                                                          *
32  *   04/30/10:                                              *
33  *                                                          *
34  *   Updated to use NadaNet 3.x boot request format.        *
35  *                                                          *
36  *   RESET code changed to force reboot if network state    *
37  *   is not ONE.  Makes power cycle much more likely to     *
38  *   cause all 'Crate machines to reboot, while still       *
39  *   permitting warm restart by resetting a machine with    *
40  *   the network held at ONE (locked).                      *
41  *                                                          *
42  *   04/18/08:                                              *
43  *                                                          *
44  *   Restricted boot code load address to $0400..$BFFF.     *
45  *                                                          *
46  *   04/15/08:                                              *
47  *                                                          *
48  *   Major modification to create minimal boot code.        *
49  *                                                          *
50  *   Changed sign-on message to "AppleCrate //e", like      *
51  *   the Enhanced //e sign-on.                              *
52  *                                                          *
53  *   Since the boot is accomplished prior to the machine    *
54  *   getting an ID, that task is left to a second stage     *
```

```
55   *   boot prefixed to the NadaNet boot image.  The need   *
56   *   to obtain an ID is communicated by setting the       *
57   *   page 3 variable 'bootself' to 0.                     *
58   *                                                        *
59   *   To provide visual indication that the machine is in  *
60   *   the "awaiting boot" state, a message is displayed    *
61   *   on the video screen and the 'send' LED is lit.       *
62   *                                                        *
63   *   06/29/05:                                            *
64   *                                                        *
65   *   Uses the new NadaNet RCVPKT routine (NadaNet 2.x)    *
66   *   with faster 8-cycle bit timings (thanks to Stephen   *
67   *   Thomas for his brilliant code).                      *
68   *                                                        *
69   *   Changed AppleCrate sign-on version to 2.0.           *
70   *                                                        *
71   *   10/20/04:                                            *
72   *                                                        *
73   *   Moved "bootself" to $3CC (out of way of ProDOS).     *
74   *                                                        *
75   *   Reorganized to use NADANET "put" files.              *
76   *                                                        *
77   *   06/23/04:                                            *
78   *                                                        *
79   *   Made "packet"/"message" nomenclature consistent.     *
80   *                                                        *
81   *   05/13/04:                                            *
82   *                                                        *
83   *   Initial ROM boot version.                            *
84   *                                                        *
85   **********************************************************
```

```
 87   ******************** Version setup ************************
 88
 89            org   $C600      ; (Enhanced //e self-test)
 90   master   equ   0          ; Non-master version
 91   dos      equ   0          ; Non-DOS version
 92   crate    equ   0          ; Non-Crate version
 93   mserve   equ   0          ; Non-Message Server version
 94   ROMboot  equ   1          ; ROM boot version
 95   enhboot  equ   1          ; Enhanced //e version
 96
 97           put   NADACONST
>1    * NadaNet Constant definitions
>2
>3    * Apple ][ definitions
>4
>5    keybd    equ   $C000      ; Keyboard port
>6    kbstrobe equ   $C010      ; Keyboard strobe
>7    VBL      equ   $C019      ; Vertical blanking
>8    spkr     equ   $C030      ; Speaker toggle
>9    an0      equ   $C058      ; Annunciator 0 base addr
>10   an1      equ   an0+2
>11   an2      equ   an0+4
>12   an3      equ   an0+6
>13   pb0      equ   $C061      ; "Pushbutton" 0 base addr
>14   pb1      equ   pb0+1
>15   pb2      equ   pb0+2
>16   ptrig    equ   $C070      ; Paddle trigger
>17   dsk6off  equ   $C0E8      ; Deselect 5.25" disk in slot 6
>18
>19   * Apple Monitor definitions
>20
>21   CSW      equ   $36        ; Output vector
>22   KSW      equ   $38        ; Input vector
>23   SOFTEV   equ   $3F2       ; Soft re-entry vector
>24   PWREDUP  equ   $3F4       ; Powered-Up check byte
>25
>26   PRBL2    equ   $F94A      ; Display (X) blanks
>27   PREAD    equ   $FB1E      ; Read PDL(X) into Y
>28   HOME     equ   $FC58      ; Clear display
>29   CROUT1   equ   $FD8B      ; Clear to EOL, then CR
>30   PRBYTE   equ   $FDDA      ; Display A as hex byte
>31   COUT     equ   $FDED      ; Display character in A
>32   BELL     equ   $FF3A      ; Beep for 100ms.
>33
>34   * Applesoft definitions
>35
>36   PSTART   equ   $67        ; Start of BASIC prog
>37   VARTAB   equ   $69        ; End prog / start vars
>38   FRETOP   equ   $6F        ; Start of string storage
>39   HIMEM    equ   $73        ; Highest BASIC mem
>40   PROGEND  equ   $AF        ; End of BASIC prog
>41   ONERR    equ   $D8        ; ONERR flag (0 = off)
```

```
>42
>43   COLDSTRT equ   $E000      ; Cold start BASIC
>44   FIXLINKS equ   $D4F2      ; Fix up BASIC prog links
>45   RUNPROG  equ   $D566      ; RUN Applesoft prog
>46
>47   * Mapping of hardware resources
>48
>49   dsend    equ   an1         ; Data 'send'
>50   drecv    equ   pb1         ; Data 'receive'
>51   zipslow  equ   dsk6off     ; Zip Chip 'slow mode' for 51 ms.
```

```
>53   * Page zero variables
>54
>55   lastidx  equ   $EB        ; Last RCVPKT buffer index
>56   ckbyte   equ   $EC        ; Check byte
>57   ptr      equ   $ED        ; Data buffer pointer (0..leng-1)
>58   address  equ   $FC        ; Scratch addr of local data
>59   length   equ   $FE        ; Scratch length of local data
>60
>61   * Protocol constants
>62
>63   cyperms  equ   1020       ; Cycles per ms. (really 1020.4)
>64
>65   arbtime  equ   1          ; Min arbitration time (ms)
>66   ]cy      equ   arbtime*cyperms ; Arbtime in cycles
>67   ]cpx     equ   11         ; Cycles per X iteration
>68   arbx     equ   ]cy/]cpx   ; X iterations
>69
>70   ]servpad equ   ]cy/4      ; Gap margin
>71   servegap equ   ]cy-]servpad/13 ; SERVER wait loop 13 cyc.
>72
>73   ]cy      equ   ]cpx*256   ; Max arb time (cycles)
>74   maxarb   equ   ]cy+cyperms/cyperms ; ceiling(max arb) (ms)
>75
>76   idletime equ   20         ; Idle polling timeout (ms)
>77                              ; (stay under 51ms for Zip Chip)
>78   reqdur   equ   6          ; Typical req duration (ms)
>79   reqpidle equ   idletime/reqdur ; Requests per idletime
>80
>81   ]cy      equ   idletime*cyperms ; Timeout in cycles
>82   ]cpx     equ   11         ; Cycles per X iteration
>83   ]cpy     equ   ]cpx*256+4 ; Cycles per Y iteration
>84   idleto   equ   ]cy/]cpy+1 ; Number of Y iterations
>85
>86   reqto    equ   1          ; Timeout within protocol is
>87                              ;  minimum arbitration time.
>88   maxgap   equ   87         ; Max intra-pkt gap (cycles)
>89   gapwait  equ   maxgap/13+1 ; MONITOR wait loop is 13 cyc.
>90
>91   reqtime  equ   3000       ; Req response timeout (ms)
>92   rqperiod equ   20         ; Milliseconds between retrys
>93   reqdelay equ   rqperiod-3 ; ARB+SEND+RCV timeout = 3ms.
>94
>95   maxreqrt equ   3          ; Max # of xxxREQ retries
>96   maxretry equ   reqtime/rqperiod/maxreqrt ; # of re-sends
```

```
 98           use   NADAMACS
>1    ****************** Macro definitions ********************
>2
>3    inc16    mac
>4             inc   ]1          ; Increment 16-bit word.
>5             do    ]1+1/$100   ; If ]1 is non-page zero
>6             bne   *+5         ; - No carry.
>7             else              ; Else if ]1 on page zero
>8             bne   *+4         ; - No carry.
>9             fin
>10            inc   ]1+1        ; Propagate carry.
>11            eom
>12
>13   mov16    mac
>14            lda   ]1          ; Move 2 bytes
>15            sta   ]2
>16            if    #=]1
>17            lda   ]1/$100     ; high byte of immediate
>18            else
>19            lda   1+]1
>20            fin
>21            sta   1+]2
>22            eom
>23
>24   delay    mac
>25            ldx   #]1/5       ; (5 cycles per iteration)
>26   ]delay   dex
>27            bne   ]delay
>28            eom
>29
>30   dlyms    mac
>31            ldy   #]1         ; Delay 1ms. per iteration
>32   ]dly     delay 1020-4      ; Cycles per ms. - 4
>33            dey
>34            bne   ]dly
>35            eom
>36
>37   align    mac
>38            ds    *-1/]1*]1+]1-*
>39            eom
>40
```

```
                 99           put   NADADEFS
                >1    ************************************************************
                >2    *                                                          *
                >3    *                  NadaNet Definitions                     *
                >4    *                        v3.1                              *
                >5    *                                                          *
                >6    *            Michael J. Mahon - Oct 13, 2004               *
                >7    *                 Revised Apr 29, 2010                     *
                >8    *                                                          *
                >9    *            Copyright (c) 2004, 2008, 2010                *
                >10   *                                                          *
                >11   ************************************************************
                >12
                >13   version  equ   $31        ; NadaNet version 3.1
                >14
                >15   ********** Control Packet Definition **********
                >16
                >17            dum   0          ; Control packet format:
0000: 00        >18   rqmd     ds    1          ;   Request & Modifier
0001: 00        >19   frmc     ds    1          ;   Complement of sending ID
0002: 00        >20   dst      ds    1          ;   Destination ID (0 = bcast)
0003: 00        >21   frm      ds    1          ;   Sending ID (never 0)
0004: 00 00     >22   adr      ds    2          ;   Address field
0006: 00 00     >23   len      ds    2          ;   Length field
                >24                             ; ========================
                >25   lenctl   ds    0          ; Length of control packet
                >26            dend
                >27
                >28   * Request codes (upper 5 bits) and modifiers (lower 3 bits)
                >29
                >30   reqfac   equ   8          ; Request code factor (2^3)
                >31   reqmask  equ   256-reqfac ; Request code mask (7..3)
                >32   modmask  equ   reqfac-1   ; Modifier code mask (2..0)
                >33
                >34            dum   reqfac     ; Request codes (0 invalid):
0008: 00 00 00  >35   r_PEEK   ds    reqfac     ;   PEEK request
0010: 00 00 00  >36   r_POKE   ds    reqfac     ;   POKE request
0018: 00 00 00  >37   r_CALL   ds    reqfac     ;   CALL request
0020: 00 00 00  >38   r_PUTMSG ds    reqfac     ;   PUTMSG request
0028: 00 00 00  >39   r_GETMSG ds    reqfac     ;   GETMSG request
0030: 00 00 00  >40   r_GETID  ds    reqfac     ;   GETID request
0038: 00 00 00  >41   r_BOOT   ds    reqfac     ;   BOOT request (in ROM)
0040: 00 00 00  >42   r_BCAST  ds    reqfac     ;   BCAST request
0048: 00 00 00  >43   r_BPOKE  ds    reqfac     ;   Broadcast POKE request
0050: 00 00 00  >44   r_PKINC  ds    reqfac     ;   PEEK & INCrement request
0058: 00 00 00  >45   r_PKPOK  ds    reqfac     ;   PEEKPOKE request
0060: 00 00 00  >46   r_RUN    ds    reqfac     ;   RUN request
0068: 00 00 00  >47   r_BRUN   ds    reqfac     ;   BRUN request
                >48                             ; ========================
                >49   maxreq   ds    0          ;   Max request + reqfac
                >50            dend
                >51
```

```
                >52             dum   1           ; Modifier codes (0 invalid):
0001: 00        >53   rm_REQ    ds    1           ;   Request
0002: 00        >54   rm_ACK    ds    1           ;   Acknowledge
0003: 00        >55   rm_DACK   ds    1           ;   Data Acknowledge
0004: 00        >56   rm_NAK    ds    1           ;   Negative Acknowledge
                >57             dend
                >58
                >59   ***************** BCAST tags ******************
                >60   *                                             *
                >61   * High byte of BCAST address field.  Tags <$D0 *
                >62   * can be confused with RAM addresses. (The low *
                >63   * byte may be an additional specification.)    *
                >64   *                                             *
                >65   ***********************************************
                >66
                >67   t_BASIC  equ   $E0         ; Applesoft BASIC program
                >68   t_SYNTH  equ   $F0         ; Crate SYNTH program
                >69   t_VOICE  equ   $F1         ; Crate SYNTH voice
                >70
                >71   ************ NadaNet Page 3 Vector *************
                >72
                >73             dum   $3CC        ; Fixed memory vector
03CC: 00        >74   bootself db    0           ; Machine ID from BOOT
03CD: 4C 00 00  >75   warmstrt jmp   0*0         ; Warm start SERVE loop entry
                >76   nadapage equ   *-1         ; NADANET load page
                >77             dend
```

```
                 100          put   NADAVECTOR
                >1    ******************** Entry Points ************************
                >2
C600: 4C FF C6  >32   entry    jmp   RESET      ; Gets control on Reset
                >34
C603: 00        >35   ROMcksum db    0*0        ; Checksum compensation byte
                >42
                 101          put   NADAVARS
                >1    ********** Parameters and variables ************
                >2
                >4             dum   $280       ; Template for ROM boot data
                >6
0280: 00        >7    self     db    0          ; Our own machine ID
0281: 00 00 00  >8    sbuf     ds    lenctl     ; Control pkt send buffer
0289: 00 00 00  >9    rbuf     ds    lenctl     ; Control pkt receive buffer
0291: 00 00     >10   locaddr  dw    0          ; Local address of req data
0293: 32        >11   retrylim db    maxretry   ; Limit of REQUEST resends
0294: 00        >12   servecnt db    0          ; SERVE iterations (0=256)
                >13
                >14   parmsiz  equ   *-self     ; Size of parameter area
                >15
                >16   ************ Counters and Version **************
                >17
0295: 5C        >18   arbxv    db    arbx       ; Arbitrate X iters (modified)
0296: 01        >19   tolim    db    reqto      ; RCVPKT timeout limit
0297: 03        >20   reqctr   db    reqpidle   ; SERVER request counter
0298: 00        >21   reqretry db    0          ; xxxREQ retries remaining
0299: 00        >22   retrycnt db    0          ; REQUEST resend count
029A: 00 00     >23   errprot  dw    0          ; Protocol error count
029C: 00 00     >24   ckerr    dw    0          ; Checksum error count
029E: 00 00     >25   frmcerr  dw    0          ; 'frmc' collision errors
02A0: 31        >26   nadaver  db    version    ; NadaNet version
                >27
                >39
                >41            dend
```

```
 102          put   SENDRCV
>1    ********************************************************
>2    *                                                      *
>3    *                LOW-LEVEL PACKET FORMAT               *
>4    *                Revised ST Jun 27, 2005               *
>5    *                                                      *
>6    * Start of packet:                                     *
>7    *                                                      *
>8    *  --//---+---//---+         +----+    +----+----+-//->  *
>9    *  Locked |  ONE   |  ZERO   |ONE |ZERO|ONE |Bit7|       *
>10   *  or Idle|  31cy  |  16cy   |8cy |8cy |8cy |8cy |       *
>11   *  --//---+        +---------+    +----+    +----+-//->  *
>12   *         |        |         |         |    |            *
>13   *         |      Start     Coarse    Servo  |<- 8 -//->  *
>14   *         |      sync       sync             |  data     *
>15   *         |                                  |  bits     *
>16   *         |                                  | (64cy)    *
>17   *         |<---- Start sequence (71cy) ---->|            *
>18   *                                                      *
>19   *  (Note: data bits are transmitted inverted - 0-bit    *
>20   *         in memory is ONE on wire and vice versa)      *
>21   *                                                      *
>22   * Interbyte separator:                                 *
>23   *                                                      *
>24   *  >-//-+----+----+              +----+----+----+-//->  *
>25   *       |Bit1|Bit0|    ZERO       |ONE |Bit7|Bit6|       *
>26   *       |8cy |8cy |   22-23cy     |8cy |8cy |8cy |       *
>27   *  >-//-+----+----+---------------+    +----+----+-//->  *
>28   *                 |               |    |               *
>29   *  >-//- 8 data ->|            Servo  |<- 8 data -//->  *
>30   *         bits    |                   |    bits         *
>31   *                 |<--- Interbyte ---->|               *
>32   *                       separator                      *
>33   *                       (30-31cy)                      *
>34   *                                                      *
>35   * Packet end:                                          *
>36   *                                                      *
>37   *  >-//-+----+----+                                    *
>38   *       |Bit1|Bit0|    ZERO (Idle)                      *
>39   *       |8cy |8cy |                                     *
>40   *  >-//-+----+----+-------------------------------//->  *
>41   *                 |                                    *
>42   *  >-//- End of ->|                                    *
>43   *       checkbyte                                      *
>44   *                                                      *
>45   ********************************************************
```

```
>287
>288  ************************************************************
>289  *                                                          *
>290  *                        R C V P K T                       *
>291  *                                                          *
>292  *             Michael J. Mahon - April 15, 1996            *
>293  *              Stephen Thomas - June 27, 2005              *
>294  *                   Revised May 21, 2008                   *
>295  *                                                          *
>296  *         Copyright (c) 1996, 2003, 2004, 2005, 2008       *
>297  *                                                          *
>298  *  Receives (X) bytes (1..256) starting at (A,Y) from      *
>299  *  the sending machine.                                    *
>300  *                                                          *
>301  *  If no packet is detected within the minimum arb time    *
>302  *  plus 'tolim'-1 times 2.8ms, it returns with carry set   *
>303  *  and A = 0.                                              *
>304  *                                                          *
>305  *  If packet is received, but checksum doesn't compare,    *
>306  *  it returns with carry set and A <> 0.                   *
>307  *                                                          *
>308  *  RCVPKT does the following steps:                        *
>309  *     1. Detect 'start signal' ONE                         *
>310  *     2. Put Zip Chip in 'slow mode' for >38,000 cycles    *
>311  *     3. Sync to cycles 5-7 of 8-cycle data cells          *
>312  *     3. Receive (X) bytes (at 93 +3/-0 cycles/byte)       *
>313  *     4. Receive check byte and verify correctness,        *
>314  *        keeping count of checksum errors.                 *
>315  *                                                          *
>316  *  RCVCTL performs a RCVPKT to the control packet          *
>317  *  receive buffer 'rbuf'.                                  *
>318  *                                                          *
>319  *  RCVPTR performs a RCVPKT to the address in 'ptr' with   *
>320  *  length (X).                                             *
>321  *                                                          *
>322  ************************************************************
>323  *                                                          *
>324  *                  Implementation Note                     *
>325  *                                                          *
>326  *  RCVPKT maintains synchronization with the data stream   *
>327  *  by using a "digital PLL" technique.  The RCVPKT byte    *
>328  *  loop is 93 cycles, which is 1 or 2 cycles shorter       *
>329  *  than the send loop.  When RCVPKT samples the servo      *
>330  *  transition and finds that it hasn't happened yet, it    *
>331  *  adds a 3-cycle delay to make the total loop time 96     *
>332  *  cycles and restore optimal sync.                        *
>333  *                                                          *
>334  *  The effect is to keep the data sampling window on the   *
>335  *  5th to 7th cycle of the 8-cycle data bitcell, in        *
>336  *  spite of the send loop buffer crossing pages at some    *
>337  *  point in a packet and clock frequency differences of    *
>338  *  +/- 1%  between sending and receiving machines.         *
```

```
>339  *                                                               *
>340  *  A similar technique assures a well-controlled sample         *
>341  *  position from the first byte of each received packet:         *
>342  *                                                               *
>343  *  After the ONE marking the packet start, there's a 16         *
>344  *  cycle ZERO.  Call the time the transmitter begins            *
>345  *  that ZERO t=0.                                               *
>346  *                                                               *
>347  *  The receive loop waits for the ZERO, sampling the            *
>348  *  bus in a tight loop with a 7-cycle period; call the          *
>349  *  time its first ZERO sample occurs rt=0.  Allowing up         *
>350  *  to 4 cycles for pulldown time on the worst network           *
>351  *  bus we can possibly work with, rt=0 could be any time        *
>352  *  between t=0 and t=11.                                        *
>353  *                                                               *
>354  *  At t=16, the transmitter will actively drive the bus         *
>355  *  to ONE (a hard-driven transition typically taking            *
>356  *  much less than 1 cycle).  At rt=10, the receive code         *
>357  *  samples the bus once again; if it sees ONE (which it         *
>358  *  will only do if rt=0 occurred between t=6 and t=11)          *
>359  *  it skips a 6-cycle time delay, arriving at rt=19 six         *
>360  *  cycles early.  This makes the rest of the timing work        *
>361  *  as if rt=0 had actually fallen between t=0 and t=5           *
>362  *  instead of t=6 and t=11.  Timings referred to rt=0           *
>363  *  now have an uncertainty of only 6 cycles with respect        *
>364  *  to t=0 instead of the 11 cycle uncertainty they began        *
>365  *  with, and the receiver is in coarse sync.                    *
>366  *                                                               *
>367  *  In the most-delayed case, with rt=0 at t=11, the             *
>368  *  rt=10 sample will occur at t=21.  Since the trans-           *
>369  *  mitter does not release the bus until t=24, this is          *
>370  *  safe.                                                        *
>371  *                                                               *
>372  *  At t=32, the transmitter will drive the bus back to          *
>373  *  ONE.  At rt=29, the receive code samples the bus and         *
>374  *  if it sees ONE (which it will only do if rt=0 fell           *
>375  *  between t=3 and t=6) it skips a 3-cycle time delay,          *
>376  *  arriving at rt=36 three cycles early.  This makes the        *
>377  *  rest of the timing work as if rt=0 actually happened         *
>378  *  between t=0 and t=3 instead of t=3 and t=6.  Timings         *
>379  *  referred to rt=0 now have an uncertainty of only 3           *
>380  *  cycles with respect to t=0, and the receiver is in           *
>381  *  fine sync.                                                   *
>382  *                                                               *
>383  *  The edge at t=32 is actually the servo edge for the          *
>384  *  first byte.  Timings within a data byte are all taken        *
>385  *  relative to the servo edge, so t=32 is redefined as          *
>386  *  t=0 and a corresponding adjustment is made to rt;            *
>387  *  the point called rt=36 in the previous paragraph is          *
>388  *  actually labelled :rt04 in the code.                         *
>389  *                                                               *
>390  *  The first data bitcell runs from t=8 to t=16.  The           *
```

```
                >391  *  receiver samples it at rt=12 - that is, some time     *
                >392  *  between t=12 and t=15.  This gives a 4-cycle margin   *
                >393  *  at the start of the bitcell and 1 cycle at the end,   *
                >394  *  which should be reliable even with truly woeful       *
                >395  *  pulldown times.                                       *
                >396  *                                                        *
                >397  *  Samples for the rest of the data bits are taken at    *
                >398  *  8-cycle intervals to match the transmit rate, and the *
                >399  *  3-cycle fine sync code is re-used to implement the    *
                >400  *  DPLL and make sure the receiver stays in sync for all *
                >401  *  subsequent data bytes.                                *
                >402  *                                                        *
                >403  **********************************************************
                >404
C604: A9 89     >405  RCVCTL   lda   #<rbuf     ; Receive control pkt to 'rbuf'
C606: A0 02     >406          ldy   #>rbuf
C608: A2 08     >407          ldx   #lenctl
C60A: 85 ED     >408  RCVPKT   sta   ptr        ; A = buf address lo
C60C: 84 EE     >409          sty   ptr+1      ; Y = buf address hi
C60E: 8A        >410  RCVPTR   txa             ; Seed checksum with length
C60F: CA        >411          dex             ; X = length 1..256 (0=>256);
C610: 86 EB     >412          stx   lastidx    ;  convert to last buffer index
                >413
C612: AC 96 02  >414          ldy   tolim      ; Wait <= (tolim-1) * 2.8ms.
C615: A2 5C     >415          ldx   #arbx      ;  plus minimum arb time.
C617: 08        >416          php             ; Save interrupt state
C618: 78        >417          sei             ;  and disable interrupts.
C619: 2C E8 C0  >418          bit   zipslow    ; Slow any Zip Chip to 1 MHz.
                >419
C61C: 2C 62 C0  >420  :waitstr bit   drecv      ; Wait for starting ONE.
C61F: 30 0A     >421          bmi   :gotstr
C621: CA        >422          dex             ; (inner loop is 11 cycles)
C622: D0 F8     >423          bne   :waitstr   ; Keep waiting...
C624: 88        >424          dey             ; (outer loop is 2820 cycles)
C625: D0 F5     >425          bne   :waitstr   ; Loop for 'timeout' ms.
                >426
C627: 28        >427          plp             ; Restore int state
C628: 98        >428          tya             ; Signal timeout (A=0, Z set)
C629: 38        >429          sec             ;  and return with C set.
C62A: 60        >430  :exit    rts
                >431
C62B: 2C E8 C0  >432  :gotstr  bit   zipslow    ; Slow Zip Chip for packet.
                >433
C62E: 2C 62 C0  >434  :waitsyn bit   drecv      ; Wait for 16-cycle sync ZERO;
C631: 30 FB     >435          bmi   :waitsyn   ;  too bad if bus locks forever!
                >436
C633: A0 FF     >437          ldy   #$FF       ; Index-1 of first data location
C635: A2 7F     >438          ldx   #$7F       ; CPX #0-7F sets C, 80-FF clears
C637: EC 62 C0  >439  :synrt07 cpx   drecv      ; Check for coarse sync at rt=10
C63A: B0 05     >440          bcs   :synrt14   ; Only do delay if still ZERO
                >441
C63C: 2C 2A C6  >442  :synrt19 bit   :exit      ; Set V (not-ckbyte flag)
```

```
C63F: 70 04    >443           bvs   :servo     ; Do first servo check at rt=29
               >444
C641: 18       >445  :synrt14 clc               ; 6-cycle coarse sync delay
C642: 90 F8    >446           bcc   :synrt19    ; (1 extra to get here, 5 back)
               >447
C644: B8       >448  :rt88    clv               ; Clear not-ckbyte flag
               >449
C645: EC 62 C0 >450  :servo   cpx   drecv       ; Check for servo transition
C648: 90 02    >451  :rt01    bcc   :rt04       ; Delay 3 cyc if past servo,
C64A: EA       >452           nop               ;  6 if not
C64B: EA       >453           nop
               >454
C64C: 85 EC    >455  :rt04    sta   ckbyte      ; Update checksum
C64E: C8       >456           iny               ; Index next data location
               >457
C64F: EC 62 C0 >458  :rt09    cpx   drecv       ; C <-- ~ bit 7 at rt=12
C652: 2A       >459           rol               ; Shift bit 7 in.
C653: EA       >460           nop
C654: EC 62 C0 >461           cpx   drecv       ; C <-- ~ bit 6 at rt=20
C657: 2A       >462           rol
C658: EA       >463           nop
C659: EC 62 C0 >464           cpx   drecv       ; C <-- ~ bit 5 at rt=28
C65C: 2A       >465           rol
C65D: EA       >466           nop
C65E: EC 62 C0 >467           cpx   drecv       ; C <-- ~ bit 4 at rt=36
C661: 2A       >468           rol
C662: EA       >469           nop
C663: EC 62 C0 >470           cpx   drecv       ; C <-- ~ bit 3 at rt=44
C666: 2A       >471           rol
C667: EA       >472           nop
C668: EC 62 C0 >473           cpx   drecv       ; C <-- ~ bit 2 at rt=52
C66B: 2A       >474           rol
C66C: EA       >475           nop
C66D: EC 62 C0 >476           cpx   drecv       ; C <-- ~ bit 1 at rt=60
C670: 2A       >477           rol
C671: EA       >478           nop
C672: EC 62 C0 >479           cpx   drecv       ; C <-- ~ bit 0 at rt=68
C675: 2A       >480  :rt69    rol
C676: 50 0A    >481  :rt71    bvc   :rcvdone    ; quit after ckbyte
               >482
C678: 91 ED    >483  :rt73    sta   (ptr),y     ; Save data (always 6cy)
C67A: 45 EC    >484  :rt79    eor   ckbyte      ; Compute checksum
C67C: C4 EB    >485  :rt82    cpy   lastidx     ; Stored last byte?
C67E: F0 C4    >486  :rt85    beq   :rt88       ; Go clear not-ckbyte flag if so
C680: D0 C3    >487  :rt87    bne   :servo      ; Do next servo sample at rt=93
               >488
C682: 45 EC    >489  :rcvdone eor   ckbyte      ; A = 0 if ckbyte = sum
C684: F0 08    >490           beq   :goodck     ; -No error.
               >491           inc16 ckerr       ; Count checksum error.
C686: EE 9C 02 >491           inc   ckerr       ; Increment 16-bit word.
C689: D0 03    >491           bne   *+5         ; - No carry.
C68B: EE 9D 02 >491           inc   ckerr+1     ; Propagate carry.
```

```
                >491            eom
C68E: 28        >492  :goodck   plp             ; Restore int state
C68F: C9 01     >493            cmp   #1        ; Set C & NZ if checksum bad,
C691: AA        >494            tax             ;  clear C and set Z if good
C692: 60        >495            rts             ;   and return.
```

```
                >497  ************************************************************
                >498  *                                                          *
                >499  *                      P R O T E R R                       *
                >500  *                                                          *
                >501  ************************************************************
                >502
                >503  PROTERR  inc16 errprot    ; Count protocol error.
C693: EE 9A 02  >503           inc   errprot    ; Increment 16-bit word.
C696: D0 03     >503           bne   *+5        ; - No carry.
C698: EE 9B 02  >503           inc   errprot+1  ; Propagate carry.
                >503           eom
C69B: 60        >504           rts
                >505
                >506
                >507  ************************************************************
                >508  *                                                          *
                >509  *                   R A R L = > A L                        *
                >510  *                                                          *
                >511  *                      R A = > A                           *
                >512  *                                                          *
                >513  ************************************************************
                >514
                >515  rarl=>al mov16 rbuf+len;length ; 'rbuf' length  -> length
C69C: AD 8F 02  >515           lda   rbuf+len   ; Move 2 bytes
C69F: 85 FE     >515           sta   length
C6A1: AD 90 02  >515           lda   1+rbuf+len
C6A4: 85 FF     >515           sta   1+length
                >515           eom
                >516  ra=>a    mov16 rbuf+adr;address; 'rbuf' address -> address
C6A6: AD 8D 02  >516           lda   rbuf+adr   ; Move 2 bytes
C6A9: 85 FC     >516           sta   address
C6AB: AD 8E 02  >516           lda   1+rbuf+adr
C6AE: 85 FD     >516           sta   1+address
                >516           eom
C6B0: 60        >517           rts
```

```
                        >562  ************************************************************
                        >563  *                                                          *
                        >564  *                      R C V L O N G                       *
                        >565  *                                                          *
                        >566  *              Michael J. Mahon - May 5, 1996              *
                        >567  *                   Revised May 21, 2008                   *
                        >568  *                                                          *
                        >569  *                 Copyright (c) 1996, 2008                 *
                        >570  *                                                          *
                        >571  *  RCVLONG receives 'length' bytes to 'address' from the   *
                        >572  *  currently sending machine.                              *
                        >573  *                                                          *
                        >574  *  It receives a series of packets if 'length' is          *
                        >575  *  greater than 256 bytes.                                 *
                        >576  *                                                          *
                        >577  *  RCVLONG detects checksum errors and timeouts, and       *
                        >578  *  returns with Carry set and A=0 if timeout, and          *
                        >579  *  Carry set and A>0 if a checksum error.  Timeouts in     *
                        >580  *  this context are protocol errors.  Both kinds of        *
                        >581  *  errors are tallyed in counters.                         *
                        >582  *                                                          *
                        >583  ************************************************************
                        >584
C6B1: A5 FF             >585  RCVLONG  lda   length+1   ; How many 256-byte pages?
C6B3: F0 11             >586          beq   :short     ; - None, just a short pkt.
C6B5: A2 00             >587  :loop    ldx   #0         ; Set 256 byte packet.
C6B7: A5 FC             >588          lda   address    ;  and point to
C6B9: A4 FD             >589          ldy   address+1  ;   data buffer.
C6BB: 20 0A C6          >590          jsr   RCVPKT     ; Receive 256 bytes.
C6BE: B0 14             >591          bcs   :err       ; Receive error detected.
C6C0: E6 FD             >592          inc   address+1  ; Advance to next page
C6C2: C6 FF             >593          dec   length+1   ;  and decrement page
C6C4: D0 EF             >594          bne   :loop      ;   count until done...
C6C6: A6 FE             >595  :short   ldx   length     ; Remaining data length.
C6C8: F0 09             >596          beq   :done      ; -All done.
C6CA: A5 FC             >597          lda   address
C6CC: A4 FD             >598          ldy   address+1
C6CE: 20 0A C6          >599          jsr   RCVPKT     ; Receive final packet.
C6D1: B0 01             >600          bcs   :err       ; Keep track of any errors.
C6D3: 60                >601  :done    rts
                        >602
C6D4: D0 03             >603  :err     bne   :ckerr     ; Checksum error.
C6D6: 20 93 C6          >604          jsr   PROTERR    ; Count protocol error.
C6D9: A8                >605  :ckerr   tay              ; Set Z flag from A.
C6DA: 60                >606          rts
```

```
                103          put   ENHBOOTCODE
               >1    ************ Enhanced Boot code definitions **************
               >2
               >3    * Boot data and definitions
               >4
C6DB: C1 F0 F0 >5    idmsg    asc   "AppleCrate //e, v3.x" ; Boot msg
               >6    idlen    equ   *-idmsg    ; Length of message
C6EF: BA A0 C1 >7             asc   ": Awaiting boot."
               >8    wtlen    equ   *-idmsg    ; Length of full msg.
```

```
                >10   ************************************************************
                >11   *                                                          *
                >12   *                        R E S E T                         *
                >13   *                                                          *
                >14   *              Michael J. Mahon - Oct 20, 2004             *
                >15   *                   Revised Apr 30, 2010                   *
                >16   *                                                          *
                >17   *                 Copyright (c) 2004, 2010                 *
                >18   *                                                          *
                >19   *  Upon power-up or hardware reset, this routine looks     *
                >20   *  at the network state.  If it is ZERO, it initiates      *
                >21   *  the boot sequence.  If it is ONE, it looks at the       *
                >22   *  $3F2 reset vector and decides whether to reboot or      *
                >23   *  warm restart.                                           *
                >24   *                                                          *
                >25   *  This normally forces a reboot, but, if the network is  *
                >26   *  held at ONE (locked), Crate machines may be reset to    *
                >27   *  force a warm restart.                                   *
                >28   *                                                          *
                >29   ************************************************************
                >30
C6FF: AD 62 C0  >31   RESET   lda   drecv      ; Sample network state
C702: 10 0D     >32           bpl   BOOT       ;  & force boot if ZERO.
C704: AD F3 03  >33           lda   SOFTEV+1   ; Is the PWREDUP byte
C707: 49 A5     >34           eor   #$A5       ;  correct for a warm
C709: CD F4 03  >35           cmp   PWREDUP    ;   restart?
C70C: D0 03     >36           bne   BOOT       ; -No, boot required.
C70E: 6C F2 03  >37           jmp   (SOFTEV)   ; -Yes, give code control.
```

```
                        >39   ********************************************************
                        >40   *                                                      *
                        >41   *                        B O O T                       *
                        >42   *                                                      *
                        >43   *            Michael J. Mahon - May 13, 2004           *
                        >44   *                 Revised Apr 30, 2010                 *
                        >45   *                                                      *
                        >46   *            Copyright (c) 2004, 2008, 2010            *
                        >47   *                                                      *
                        >48   *  Boot a diskless machine from the network.           *
                        >49   *                                                      *
                        >50   *  This version of BOOT is passive, to allow it to be  *
                        >51   *  shorter than $200.  This allows it to fit in the ROM *
                        >52   *  of an Enhanced //e ROM, replacing the self-test code *
                        >53   *  from $C600..C7FF.                                   *
                        >54   *                                                      *
                        >55   *  BOOT does the following steps:                      *
                        >56   *     1. Clear screen and display "Awaiting boot".     *
                        >57   *     2. Go into a RCVCTL loop waiting for the broadcast *
                        >58   *        BOOT request that contains the load address   *
                        >59   *        ($0400..$BFFF) and length of the boot code.   *
                        >60   *     3. Receive the boot code into memory.            *
                        >61   *     4. If any error, retry starting at step 1.       *
                        >62   *     5. Clear screen and display ID message.          *
                        >63   *     6. Jump to start of second-stage boot.           *
                        >64   *                                                      *
                        >65   ********************************************************
                        >66
C711: 20 58 FC          >67   BOOT     jsr   HOME       ; Clear the display
C714: A2 01             >68           ldx   #40-wtlen/2-1 ; and print enough
C716: 20 4A F9          >69           jsr   PRBL2      ;     blanks to center
C719: A0 00             >70           ldy   #0         ;      the "Awaiting boot"
C71B: B9 DB C6          >71   :msglp   lda   idmsg,y    ;       message.
C71E: 20 ED FD          >72           jsr   COUT
C721: C8                >73           iny
C722: C0 24             >74           cpy   #wtlen
C724: 90 F5             >75           bcc   :msglp
C726: A2 3A             >76   :restart ldx   #servegap  ; Delay min arb time.
C728: CD E8 C0          >77           cmp   zipslow    ; Zip Chip to 1MHz mode.
C72B: AC 62 C0          >78           ldy   drecv      ; Sample net state.
C72E: 98                >79   :waitidl tya
C72F: 4D 62 C0          >80           eor   drecv      ; Has net changed?
C732: 30 F2             >81           bmi   :restart   ; -Yes, restart timing.
C734: CA                >82           dex              ; -No, count down.
C735: D0 F7             >83           bne   :waitidl
C737: A9 08             >84           lda   #idleto    ; Set RCVPKT timeout
C739: 8D 96 02          >85           sta   tolim      ;  to 20ms.
C73C: AD 70 C0          >86   :again   lda   ptrig      ; Flash "Send" LED while
C73F: 20 04 C6          >87           jsr   RCVCTL     ;  waiting for a BOOT req.
C742: 90 04             >88           bcc   :ok        ; -good pkt.
C744: F0 F6             >89           beq   :again     ; -timed out, go again.
C746: D0 DE             >90           bne   :restart   ; -bad cksum, start over.
```

```
                >91
C748: AD 8B 02  >92    :ok      lda   rbuf+dst   ; Broadcast?
C74B: D0 D9     >93             bne   :restart   ; -No, start over.
C74D: AD 8A 02  >94             lda   rbuf+frmc  ; Verify that
C750: 49 FF     >95             eor   #$FF       ;  complement of 'frmc'
C752: CD 8C 02  >96             cmp   rbuf+frm   ;   is equal to 'frm'.
C755: D0 CF     >97             bne   :restart   ; -No, start over.
C757: AD 89 02  >98             lda   rbuf+rqmd
C75A: C9 39     >99             cmp   #r_BOOT+rm_REQ ; BOOT request?
C75C: D0 C8     >100            bne   :restart   ; -No, start over.
C75E: A9 01     >101            lda   #reqto     ; Set RCVPKT timeout
C760: 8D 96 02  >102            sta   tolim      ;  short again.
C763: 20 9C C6  >103            jsr   rarl=>al   ; Set address/length.
C766: AD 8E 02  >104            lda   rbuf+adr+1 ; Look at adr hi byte
C769: C9 04     >105            cmp   #$04       ; Load addr < $0400
C76B: 90 B9     >106            bcc   :restart   ;  not permitted.
C76D: C9 C0     >107            cmp   #$C0       ; Load addr >= $C000
C76F: B0 B5     >108            bcs   :restart   ;  not permitted.
C771: 20 B1 C6  >109            jsr   RCVLONG    ; Read the boot code.
C774: B0 B0     >110            bcs   :restart   ; -error, start over.
C776: 20 58 FC  >111            jsr   HOME       ; Clear the display
C779: A2 09     >112            ldx   #40-idlen/2-1 ; and print enough
C77B: 20 4A F9  >113            jsr   PRBL2      ;     blanks to center
C77E: A0 00     >114            ldy   #0         ;      the banner.
C780: B9 DB C6  >115   :msgloop lda   idmsg,y
C783: 20 ED FD  >116            jsr   COUT
C786: C8        >117            iny
C787: C0 14     >118            cpy   #idlen
C789: 90 F5     >119            bcc   :msgloop
C78B: A9 00     >120            lda   #0         ; Set bootself to 0 to run
C78D: 8D CC 03  >121            sta   bootself   ;  second-stage boot code.
C790: 6C 8D 02  >122            jmp   (rbuf+adr) ; Give boot code control...
```

```
                104           align 256        ; Align to page boundary
C793: 00 00 00  104           ds    *-1/256*256+256-*
                104           eom
                105  endcode  equ   *
                106           err   *-1/$C800  ; Can't exceed $C800


--End assembly, 512 bytes, Errors: 0


Symbol table - alphabetical order:

?  BELL    =$FF3A      BOOT    =$C711   ?  COLDSTRT=$E000      COUT    =$FDED
?  CROUT1  =$FD8B   ?  CSW     =$36     ?  FIXLINKS=$D4F2   ?  FRETOP  =$6F
?  HIMEM   =$73        HOME    =$FC58   ?  KSW     =$38     ?  ONERR   =$D8
   PRBL2   =$F94A   ?  PRBYTE  =$FDDA   ?  PREAD   =$FB1E   ?  PROGEND =$AF
   PROTERR =$C693   ?  PSTART  =$67        PWREDUP =$03F4      RCVCTL  =$C604
   RCVLONG =$C6B1      RCVPKT  =$C60A   ?  RCVPTR  =$C60E      RESET   =$C6FF
   ROMboot =$01     ?  ROMcksum=$C603   ?  RUNPROG =$D566      SOFTEV  =$03F2
?  VARTAB  =$69     ?  VBL     =$C019   V  ]cpx    =$0B     V  ]cpy    =$0B04
V  ]cy     =$4FB0   V  ]servpad=$FF        address =$FC        adr     =$04
MD align   =$8000      an0     =$C058      an1     =$C05A   ?  an2     =$C05C
?  an3     =$C05E      arbtime =$01        arbx    =$5C     ?  arbxv   =$0295
   bootself=$03CC      ckbyte  =$EC        ckerr   =$029C      crate   =$00
   cyperms =$03FC   MD?delay   =$8000   MD?dlyms   =$8000   ?  dos     =$00
   drecv   =$C062   ?  dsend   =$C05A      dsk6off =$C0E8      dst     =$02
?  endcode =$C800      enhboot =$01     ?  entry   =$C600      errprot =$029A
   frm     =$03        frmc    =$01     ?  frmcerr =$029E   ?  gapwait =$07
   idlen   =$14        idletime=$14        idleto  =$08        idmsg   =$C6DB
MD inc16   =$8000   ?  kbstrobe=$C010   ?  keybd   =$C000      lastidx =$EB
   len     =$06        lenctl  =$08        length  =$FE     ?  locaddr =$0291
   master  =$00     ?  maxarb  =$03        maxgap  =$57     ?  maxreq  =$70
   maxreqrt=$03        maxretry=$32     ?  modmask =$07     MD mov16   =$8000
?  mserve  =$00     ?  nadapage=$03CF   ?  nadaver =$02A0   ?  parmsiz =$15
   pb0     =$C061      pb1     =$C062   ?  pb2     =$C063      ptr     =$ED
   ptrig   =$C070   ?  r_BCAST =$40        r_BOOT  =$38     ?  r_BPOKE =$48
?  r_BRUN  =$68     ?  r_CALL  =$18     ?  r_GETID =$30     ?  r_GETMSG=$28
?  r_PEEK  =$08     ?  r_PKINC =$50     ?  r_PKPOK =$58     ?  r_POKE  =$10
?  r_PUTMSG=$20     ?  r_RUN   =$60     ?  ra=>a   =$C6A6      rarl=>al=$C69C
   rbuf    =$0289   ?  reqctr  =$0297   ?  reqdelay=$11      reqdur  =$06
   reqfac  =$08     ?  reqmask =$F8        reqpidle=$03     ?  reqretry=$0298
   reqtime =$0BB8      reqto   =$01     ?  retrycnt=$0299   ?  retrylim=$0293
?  rm_ACK  =$02     ?  rm_DACK =$03     ?  rm_NAK  =$04        rm_REQ  =$01
   rqmd    =$00        rqperiod=$14     ?  sbuf    =$0281      self    =$0280
?  servecnt=$0294      servegap=$3A     ?  spkr    =$C030   ?  t_BASIC =$E0
?  t_SYNTH =$F0     ?  t_VOICE =$F1        tolim   =$0296      version =$31
?  warmstrt=$03CD      wtlen   =$24        zipslow =$C0E8


Symbol table - numerical order:

   master  =$00     ?  dos     =$00        crate   =$00     ?  mserve  =$00
```

```
   rqmd    =$00        ROMboot =$01        enhboot =$01        arbtime =$01
   reqto   =$01        frmc    =$01        rm_REQ  =$01        dst     =$02
?  rm_ACK  =$02     ?  maxarb  =$03        reqpidle=$03        maxreqrt=$03
   frm     =$03     ?  rm_DACK =$03        adr     =$04     ?  rm_NAK  =$04
   reqdur  =$06        len     =$06     ?  gapwait =$07     ?  modmask =$07
   idleto  =$08        lenctl  =$08        reqfac  =$08     ?  r_PEEK  =$08
V  ]cpx    =$0B     ?  r_POKE  =$10     ?  reqdelay=$11        idletime=$14
   rqperiod=$14        idlen   =$14     ?  parmsiz =$15     ?  r_CALL  =$18
?  r_PUTMSG=$20        wtlen   =$24     ?  r_GETMSG=$28     ?  r_GETID =$30
   version =$31        maxretry=$32     ?  CSW     =$36     ?  KSW     =$38
   r_BOOT  =$38        servegap=$3A     ?  r_BCAST =$40     ?  r_BPOKE =$48
?  r_PKINC =$50        maxgap  =$57     ?  r_PKPOK =$58        arbx    =$5C
?  r_RUN   =$60     ?  PSTART  =$67     ?  r_BRUN  =$68     ?  VARTAB  =$69
?  FRETOP  =$6F     ?  maxreq  =$70     ?  HIMEM   =$73     ?  PROGEND =$AF
?  ONERR   =$D8     ?  t_BASIC =$E0        lastidx =$EB        ckbyte  =$EC
   ptr     =$ED     ?  t_SYNTH =$F0     ?  t_VOICE =$F1     ?  reqmask =$F8
   address =$FC        length  =$FE     V  ]servpad=$FF        self    =$0280
?  sbuf    =$0281      rbuf    =$0289   ?  locaddr =$0291   ?  retrylim=$0293
?  servecnt=$0294   ?  arbxv   =$0295      tolim   =$0296   ?  reqctr  =$0297
?  reqretry=$0298   ?  retrycnt=$0299      errprot =$029A      ckerr   =$029C
?  frmcerr =$029E   ?  nadaver =$02A0      bootself=$03CC   ?  warmstrt=$03CD
?  nadapage=$03CF      SOFTEV  =$03F2      PWREDUP =$03F4      cyperms =$03FC
V  ]cpy    =$0B04      reqtime =$0BB8   V  ]cy     =$4FB0   MD align   =$8000
MD?dlyms   =$8000   MD?delay   =$8000   MD mov16   =$8000   MD inc16   =$8000
?  keybd   =$C000   ?  kbstrobe=$C010   ?  VBL     =$C019   ?  spkr    =$C030
   an0     =$C058      an1     =$C05A   ?  dsend   =$C05A   ?  an2     =$C05C
?  an3     =$C05E      pb0     =$C061      pb1     =$C062      drecv   =$C062
?  pb2     =$C063      ptrig   =$C070      dsk6off =$C0E8      zipslow =$C0E8
?  entry   =$C600   ?  ROMcksum=$C603      RCVCTL  =$C604      RCVPKT  =$C60A
?  RCVPTR  =$C60E      PROTERR =$C693      rarl=>al=$C69C   ?  ra=>a   =$C6A6
   RCVLONG =$C6B1      idmsg   =$C6DB      RESET   =$C6FF      BOOT    =$C711
?  endcode =$C800   ?  FIXLINKS=$D4F2   ?  RUNPROG =$D566   ?  COLDSTRT=$E000
   PRBL2   =$F94A   ?  PREAD   =$FB1E      HOME    =$FC58   ?  CROUT1  =$FD8B
?  PRBYTE  =$FDDA      COUT    =$FDED   ?  BELL    =$FF3A
```