```
1   ************************************************************
2   *                                                          *
3   *                    N A D A . C R A T E                   *
4   *                                                          *
5   *              AppleCrate version of NadaNet               *
6   *                                                          *
7   *            Michael J. Mahon - April 14, 1996             *
8   *                  Revised May 3, 2010                     *
9   *                                                          *
10  * Copyright (c) 1996, 2003, 2004, 2005, 2008, 2009, 2010 *
11  *                                                          *
12  *   NadaNet is a suite of 6502 machine code routines       *
13  *   to support bidirectional communication among several   *
14  *   Apple // computers.  It uses one wire (plus ground)    *
15  *   connected to the game ports of the machines.           *
16  *                                                          *
17  *   An annunciator output is used to "broadcast" to all   *
18  *   machines, and a "pushbutton" input is used to sense    *
19  *   the state of the shared signalling wire.  This is      *
20  *   similar to Ethernet, but at lower speed and at TTL     *
21  *   levels.                                                *
22  *                                                          *
23  *   The raw signaling speed is 1 bit every 8 cycles, or    *
24  *   127.6 kilobaud.  With byte separator overhead of 31    *
25  *   cycles, this translates to 1 byte every 94-95 cycles,  *
26  *   or over 10K bytes/sec (thanks to Stephen Thomas!).     *
27  *                                                          *
28  *   All signal transmission and reception is done with     *
29  *   precisely timed software routines.  Synchronization    *
30  *   is assured by a digital PLL at the receiver which      *
31  *   adapts to variations in timing of 93-96 cycles/byte.   *
32  *   (If a machine has a Zip Chip accelerator installed,    *
33  *   it is temporarily slowed during packet transmission    *
34  *   and reception.)                                        *
35  *                                                          *
36  ************************************************************
37
38  ******************** Version setup ************************
39
40  SIZE     equ   $800       ; "Do not exceed" size
41
42           org   $C000-SIZE-$100 ; AppleCrate & boot page
43  master   equ   0           ; Omit master-only functions
44  dos      equ   0           ; Non-DOS version
45  crate    equ   1           ; AppleCrate version
46  mserve   equ   0           ; Non-Message Server version
47  ROMboot  equ   0           ; Non-ROM version
48  enhboot  equ   0           ; Non-Enhanced //e ROM version
```

```
 50          put   NADAHIST
>1    ************************************************************
>2    *                                                          *
>3    *                     Change History                       *
>4    *                                                          *
>5    *   05/03/10:                                              *
>6    *                                                          *
>7    *   Fixed &TIMEOUT to return to AMPERSAND to set error     *
>8    *   flags (0) and (1) properly.                            *
>9    *                                                          *
>10   *   04/28/10:                                              *
>11   *                                                          *
>12   *   Removed version 2.x code from BOOTREQ.                 *
>13   *                                                          *
>14   *   Added PEEKPOKE request/server for atomic semaphores.   *
>15   *                                                          *
>16   *   Changed JSR to service routine in AMPERSAND to go      *
>17   *   through an indirect JMP to avoid code modification.    *
>18   *                                                          *
>19   *   Factored out error retry code for simple requests      *
>20   *   (SIMPLREQ) to save space in CALLREQ.                   *
>21   *                                                          *
>22   *   02/10/09 === Released NadaNet 3.0                      *
>23   *                                                          *
>24   *   01/24/09:                                              *
>25   *                                                          *
>26   *   Added ONERR ($D8) definition and code to clear the     *
>27   *   flag at boot time and at &RUN time.                    *
>28   *                                                          *
>29   *   01/06/09:                                              *
>30   *                                                          *
>31   *   Modified NADA.CRATE's NADABOOT2 to coldstart BASIC     *
>32   *   and set HIMEM to base of NadaNet.                      *
>33   *                                                          *
>34   *   Changed 'version' to a 2-digit BCD value that is       *
>35   *   used in messages and the version byte.                 *
>36   *                                                          *
>37   *   11/11/08:                                              *
>38   *                                                          *
>39   *   Modified RUNSRV to save and restore CSW/KSW hooks      *
>40   *   so that &RUN works properly with or without an OS.     *
>41   *                                                          *
>42   *   11/03/08:                                              *
>43   *                                                          *
>44   *   Added call to FIXLINKS into RUNSRV code to allow       *
>45   *   BASIC programs to be &RUN at any address > $800.       *
>46   *                                                          *
>47   *   10/06/08:                                              *
>48   *                                                          *
>49   *   Added simple BCAST action to SERVER that just sets     *
>50   *   'address' and 'length' to request values and then      *
>51   *   returns to the calling code to deal with the data.     *
```

```
>52   *                                                               *
>53   *   Added table of BCAST tags to NADADEFS to serve as a         *
>54   *   central directory of BCAST tag values.                      *
>55   *                                                               *
>56   *   09/25/08:                                                   *
>57   *                                                               *
>58   *   Added &RUN and &BRUN, as derivatives of &POKE, to           *
>59   *   run Applesoft programs and M/L programs.                    *
>60   *                                                               *
>61   *   Added RCVCTL, RCVPTR, rarl=>al, and RCVLONG to the          *
>62   *   entry point vector for use by BCAST server code.            *
>63   *                                                               *
>64   *   09/04/08:                                                   *
>65   *                                                               *
>66   *   Restructured SERVER to correct failure to re-sync if        *
>67   *   'reqctr' was satisfied, and to minimze "deaf" time          *
>68   *   when iterating in SERVER.                                   *
>69   *                                                               *
>70   *   Added 'reqpidle' (requests per idletime) definition,        *
>71   *   which is closer to typical.                                 *
>72   *                                                               *
>73   *   08/20/08:                                                   *
>74   *                                                               *
>75   *   Added BCAST request as a general mechanism for              *
>76   *   broadcasting data.                                          *
>77   *                                                               *
>78   *   Added BCASTARB to arbitrate and lock net, then delay        *
>79   *   for 20ms. to allow all collisions to resolve and any        *
>80   *   "slow" pollers to get into their RCVCTL holds.  This        *
>81   *   arbitration will precede all broadcast requests, like       *
>82   *   BOOT, BCAST, and BPOKE.                                     *
>83   *                                                               *
>84   *   08/16/08:                                                   *
>85   *                                                               *
>86   *   Changed control packet format:                              *
>87   *   - Combined 'req' and 'mod' bytes into 'rqmd' byte.          *
>88   *   - Added complement of 'frm', called 'frmc', as a way        *
>89   *     to detect collisions of synchronized packets.             *
>90   *   - Removed "delayed BOOTREQ after GETID" boot protocol       *
>91   *   - Modified BOOTREQ to send old format packet for            *
>92   *     compatibility with v2.1 PassiveBoot ROM.                  *
>93   *   - Changed sign-on version to v3.0.                          *
>94   *                                                               *
>95   *   Moved error counters to just after IDTBL, and               *
>96   *   prefixed them with NadaNet version in hex.                  *
>97   *                                                               *
>98   *   07/25/08 === Released NadaNet 2.1                           *
>99   *                                                               *
>100  *   05/21/08:                                                   *
>101  *                                                               *
>102  *   Made numerous significant space optimizations:              *
>103  *   - Added subroutines to set 'address' & 'length'             *
```

```
>104  *     from most common variables.                          *
>105  *   - Added PROTERR subroutine to increment count.         *
>106  *   - Put checksum counting inside RCVPKT.                 *
>107  *   - Added variable delay preceding SENDLONG.             *
>108  *   - Placed all of the above in "slack" space following   *
>109  *     page-aligned SENDPKT and RCVPKT.                     *
>110  *   - Deleted MONITOR (can always load when needed).       *
>111  *                                                          *
>112  *   Fixed potential bug if INSTALL called >255 times.      *
>113  *                                                          *
>114  *   Added ID error checking to 'setid' and INIT.           *
>115  *                                                          *
>116  *   Changed broadcast BOOTREQ to not be protocol error.    *
>117  *                                                          *
>118  *   Added 'xmain', 'xsend', and 'xreceive' symbols to      *
>119  *   make slack space easy to read in symbol table.         *
>120  *                                                          *
>121  *   04/21/08:                                              *
>122  *                                                          *
>123  *   Added 'svrxkbd' entry to SERVER, used by 'servelp'     *
>124  *   to ignore keyboard input.                              *
>125  *                                                          *
>126  *   04/15/08:                                              *
>127  *                                                          *
>128  *   Split NADADEFS include file into three parts so that   *
>129  *   the control packet definitions could be used without   *
>130  *   generating code for the vectors and variables.         *
>131  *                                                          *
>132  *   Added new "Enhanced boot" protocol for AppleCrate      *
>133  *   machines using Enhanced //e's.  The new protocol       *
>134  *   blindly broadcasts the boot code whenever &BOOTCODE    *
>135  *   is invoked.                                            *
>136  *                                                          *
>137  *   Since only RCVPKT and RCVLONG, plus RESET and the      *
>138  *   actual boot logic need reside in ROM, it fits easily   *
>139  *   into the 2 pages of code used by the Enhanced //e's    *
>140  *   self-test code.                                        *
>141  *                                                          *
>142  *   To support this, a new conditional assembly flag,      *
>143  *   'enhboot' has been added and used to select the        *
>144  *   code in SENDRCV and NADADEFS for the new boot ROM.     *
>145  *                                                          *
>146  *   The new AppleCrate boot image will be prefixed with    *
>147  *   a second-stage boot that will use GETID to allocate    *
>148  *   unique machine IDs.  The DACK length hi-byte sent by   *
>149  *   Enhanced machines contains a "magic number" ($A5) to   *
>150  *   signal that GETID need not schedule a boot code send.  *
>151  *                                                          *
>152  *   The second stage boot will set up the page 3 RESET     *
>153  *   vector to go directly to NadaNet INIT, so it does      *
>154  *   not take up space in the running machine.              *
>155  *                                                          *
```

```
>156  *   The second-stage boot code uses a non-zero 'bootself'*
>157  *   value to indicate that a GETID has already been done *
>158  *   and the second-stage can be skipped for unenhanced   *
>159  *   AppleCrate machines.                                 *
>160  *                                                        *
>161  *   The maximum number of machines has been increased to *
>162  *   31, and the zeroth entry in the IDTABLE is 31.       *
>163  *                                                        *
>164  *   07/21/05 === Released NadaNet 2.0                    *
>165  *                                                        *
>166  *   06/29/05:                                            *
>167  *                                                        *
>168  *   Added NadaNet version 2.0 sign-on message to INIT.   *
>169  *                                                        *
>170  *   Replaced tree-like data send routing in SENDPKT with *
>171  *   new, shorter, lattice-like routine created by Stephen*
>172  *   Thomas.  The new routine sends all bits in uniform   *
>173  *   cells of 8 cycles, lowering the cycles/byte to 95    *
>174  *   from 106, for a speed increase of more than 11%.     *
>175  *                                                        *
>176  *   The new data transfer rate is over 10 KB/second.     *
>177  *                                                        *
>178  *   The packet start synchronization now allows RCVPKT   *
>179  *   to establish fine sync for the first byte.           *
>180  *                                                        *
>181  *   The new RCVPKT samples data bitcells only during the *
>182  *   5th, 6th, and 7th of 8 cycles, making NadaNet much   *
>183  *   more tolerant of long network time constants caused  *
>184  *   by too much cable or too high a pulldown resistance. *
>185  *                                                        *
>186  *   The timing of the check byte is now identical to the *
>187  *   timing of data bytes.                                *
>188  *                                                        *
>189  *   RCVPKT is also changed to reflect the new timings,   *
>190  *   which required unrolling the receive code again.     *
>191  *                                                        *
>192  *   Increased receive-to-send turnaround delays in       *
>193  *   PEEKSRV and GETMSRV to allow some margin for the     *
>194  *   receiving machine to start polling.                  *
>195  *                                                        *
>196  *   06/08/05:                                            *
>197  *                                                        *
>198  *   Fixed bus fight in RCVPKT pointed out by an astute   *
>199  *   reader of the code, Stephen Thomas, who also sent    *
>200  *   replacement code which only reads the paddle input   *
>201  *   and which cleverly combines data shifting with loop  *
>202  *   control, permitting the receive code to be re-rolled *
>203  *   to save space!                                       *
>204  *                                                        *
>205  *   12/05/04 === Released NadaNet 1.2                    *
>206  *                                                        *
>207  *   12/01/04:                                            *
```

```
>208  *                                                               *
>209  *   Fixed GETID bug that left 'sbuf+adr' unset if the ID        *
>210  *   received was not a temporary ID.  (For masters only)        *
>211  *                                                               *
>212  *   Parameterized maximum number of machines (maxid) and        *
>213  *   changed GETID so that any ID > maxid is considered a        *
>214  *   temporary ID to be assigned a permanent ID.                 *
>215  *                                                               *
>216  *   Changed handling of protocol errors in SERVER so            *
>217  *   they are "timed" as if they were requests.                  *
>218  *                                                               *
>219  *   11/17/04:                                                   *
>220  *                                                               *
>221  *   Changed 'servegap' wait time to 3/4 of min arb time         *
>222  *   to allow some margin for server routine processing           *
>223  *   after network is released (which is subtracted from         *
>224  *   "SERVER visible" inter-request gap).                        *
>225  *                                                               *
>226  *   11/13/04                                                    *
>227  *                                                               *
>228  *   Changed AmperNada handler to leave return variable          *
>229  *   unchanged when an error occurs.                             *
>230  *                                                               *
>231  *   11/12/04:                                                   *
>232  *                                                               *
>233  *   Increased BPOKE locked wait time to 20ms. to allow          *
>234  *   more "dead time" in an Applesoft &SERVE polling loop.       *
>235  *                                                               *
>236  *   11/10/04:                                                   *
>237  *                                                               *
>238  *   Changed SERVER gap wait to wait for an unchanging           *
>239  *   net, not an idle net, so that a BPOKE is received           *
>240  *   when preceded by a locked state.                            *
>241  *                                                               *
>242  *   Fixed bug in PKINCSRV that dropped carry.                   *
>243  *                                                               *
>244  *   11/08/04:                                                   *
>245  *                                                               *
>246  *   Changed AmperNada handler to throw an Applesoft             *
>247  *   "DATA" (49) error by default when a command fails.          *
>248  *   This error can be caught by an active ONERR, or it          *
>249  *   can be suppressed by appending a "#" to the command.        *
>250  *   If the error is suppressed, it is the programmer's          *
>251  *   responsibility to check status by PEEKing 1 and 0.          *
>252  *                                                               *
>253  *   11/06/04:                                                   *
>254  *                                                               *
>255  *   Removed &ONERR(err?) because its residual effects--         *
>256  *   storing status into variable memory--outlast any            *
>257  *   running program unless explicitly cancelled.  Using         *
>258  *   PEEK(1) is a safe and effective alternative solution.       *
>259  *                                                               *
```

```
>260  *   11/05/04:                                                *
>261  *                                                            *
>262  *   Added BPOKE & PEEKINC requestors and servers.            *
>263  *                                                            *
>264  *   Added &IDTBL(val?) to retrieve address of 'idtable'      *
>265  *   in 'master' version.                                     *
>266  *                                                            *
>267  *   Changed ARBTRATE to use a single loop, and SETID to      *
>268  *   use ID for 'arbxv' when a temp ID (>$7F) is used.        *
>269  *                                                            *
>270  *   11/01/04:                                                *
>271  *                                                            *
>272  *   Integrated AmperNada ampersand interface for BASIC       *
>273  *   into NadaNet.  Size limit is now $900.                   *
>274  *                                                            *
>275  *   10/27/04:                                                *
>276  *                                                            *
>277  *   Fixed latent BOOT timing bug in server.                  *
>278  *                                                            *
>279  *   Changed SERVER so that it returns after processing       *
>280  *   any request, in addition to when a key is pressed.       *
>281  *                                                            *
>282  *   Changed SERVER and CALLSRV to do indirect jumps,         *
>283  *   rather than pushing addresses on stack for rts.          *
>284  *                                                            *
>285  *   Changed REQUEST resend count so that the request         *
>286  *   timeout set by 'reqtime' is accurate.                    *
>287  *                                                            *
>288  *   Changed MONITOR to wait for a minimum period of          *
>289  *   unchanging network state, rather than '0' state, so      *
>290  *   a locked state between packets is detected as a gap.     *
>291  *                                                            *
>292  *   10/20/04:                                                *
>293  *                                                            *
>294  *   Added changes so that NADABOOT could be built using      *
>295  *   standard NADANET "put" files.                            *
>296  *                                                            *
>297  *   Split this change history into a separate file.          *
>298  *                                                            *
>299  *   10/18/04:                                                *
>300  *                                                            *
>301  *   Added code to INIT to set up $3CD with warm start        *
>302  *   'JMP servelp', so $3CF is NADANET's load page.           *
>303  *                                                            *
>304  *   10/13/04:                                                *
>305  *                                                            *
>306  *   Made PUTMREQ and GETMREQ conditional upon 'master'       *
>307  *   conditional compile flag.  PUTMSRV and GETMSRV are       *
>308  *   conditional upon 'not master'.  This frees up space      *
>309  *   for additional enhancements by splitting NadaNet into    *
>310  *   different functional subsets for different purposes.     *
>311  *                                                            *
```

```
>312  *   Made RCVPKT timeout variable so it can be set to the *
>313  *   minimum arbitration time while within a protocol,    *
>314  *   to protect the protocol from "outside" interference, *
>315  *   and set to 20ms. outside a protocol, when SERVE or   *
>316  *   MONITOR is running, to reduce polling dead time.     *
>317  *                                                        *
>318  *   Moved packet-starting 'ONE' earlier in SENDPKT so    *
>319  *   that ARBTRATE and RCVPKT do not need to double-poll  *
>320  *   bus to detect start pulse.                           *
>321  *                                                        *
>322  *   Changed BOOTREQ to use boot code address, length,    *
>323  *   and local address set up prior to SERVER call.       *
>324  *                                                        *
>325  *   Split entry point vector and variable definitions    *
>326  *   out into NADADEFS "put" file for use in other progs. *
>327  *                                                        *
>328  *   10/05/04:                                            *
>329  *                                                        *
>330  *   Changed ARBTRATE to lock the bus after a successful  *
>331  *   poll, so that the arbitration "increment" could be   *
>332  *   reduced to 22 cycles from 66.                        *
>333  *                                                        *
>334  *   Changed SETID arbitration time calculation to match. *
>335  *                                                        *
>336  *   Changed SERVER so that received requests, whether    *
>337  *   acted upon or not, are counted as 1/8 of a 20ms.     *
>338  *   timeout interval.  This allows time-related events   *
>339  *   to occur properly whether the net is busy or idle.   *
>340  *                                                        *
>341  *   Moved 'sbuf', 'rbuf', and counters so that they will *
>342  *   move less in the future.                             *
>343  *                                                        *
>344  *   Made REQUEST retry limit an initialized variable so  *
>345  *   that it can be lowered to speed up detection of a    *
>346  *   possibly non-existent machine.                       *
>347  *                                                        *
>348  *   Moved the 'monch' table used by PUTMSRV and GETMSRV  *
>349  *   from internal memory to the unused top of the page   *
>350  *   map table, saving 48 bytes of program memory.        *
>351  *                                                        *
>352  *   07/26/04 === Released NadaNet 1.1                    *
>353  *                                                        *
>354  *   06/23/04:                                            *
>355  *                                                        *
>356  *   Added iteration counter to SERVER and dispensed with *
>357  *   SERVE1.  Changed SERVE sync wait to min arbitration  *
>358  *   time.                                                *
>359  *                                                        *
>360  *   Disabled interrupts during SENDPKT and RCVPKT.       *
>361  *                                                        *
>362  *   Made "packet"/"message" nomenclature consistent.     *
>363  *                                                        *
```

```
>364  *   06/04/04:                                                  *
>365  *                                                              *
>366  *   Made INIT, SERVE, and BOOT functions and PEEK, POKE,       *
>367  *   and CALL functions separate include modules.               *
>368  *                                                              *
>369  *   Added text graphics for protocols.                         *
>370  *                                                              *
>371  *   05/26/04                                                   *
>372  *                                                              *
>373  *   Added MONITOR function for snooping all packets on         *
>374  *   the network and logging the first 8 bytes in memory.       *
>375  *                                                              *
>376  *   05/15/04:                                                  *
>377  *                                                              *
>378  *   Added "master" conditional assembly switch to              *
>379  *   control newly added  boot functions, BOOTREQ and           *
>380  *   GETIDSRV.                                                  *
>381  *                                                              *
>382  *   05/06/04:                                                  *
>383  *                                                              *
>384  *   Shortened arbitration time to 1 ms., since almost          *
>385  *   all protocols have less than 1 ms. delay between           *
>386  *   packets.  Exceptions will "lock" the net by pulling        *
>387  *   it high until they can respond.  This is effectively       *
>388  *   an extended "start" pulse, and RCVPKT will wait            *
>389  *   indefinitely for the transition to low.                    *
>390  *                                                              *
>391  *   Currently, only PUTMSRV and GETMSRV can take longer        *
>392  *   than 1 ms. to respond with ACK or NAK, so they must        *
>393  *   lock the net until their response.                         *
>394  *                                                              *
>395  *   The delay from last arbitration poll until beginning       *
>396  *   of "start" pulse is 54 cycles, so to avoid collision       *
>397  *   the arbitration delay difference between machines          *
>398  *   must exceed 54.  Since it must be a multiple of 11         *
>399  *   cycles, the offset is machine ID * 66 cycles.              *
>400  *                                                              *
>401  *   Since 8-byte data packets are indistinguishable            *
>402  *   from control packets, and since data packets are           *
>403  *   never delayed from a preceding control packet by           *
>404  *   more than 0.5 ms., SERVER must wait for a net "idle"       *
>405  *   (low) state for 0.5 ms. to ensure that the next pkt        *
>406  *   it receives is not data.  This delay is only needed        *
>407  *   if the net has not been polled for more than the           *
>408  *   arbitration delay (~1 ms.).                                *
>409  *                                                              *
>410  *   04/19/04:                                                  *
>411  *                                                              *
>412  *   Changed RCVPKT timeout to 20 ms., since responses          *
>413  *   are expected in much less.  SERVER loop results in         *
>414  *   "blind" time of less than 0.04 ms. per iteration.          *
>415  *                                                              *
```

```
>416  *   Changed CALLSRV to pass parameters A and X passed    *
>417  *   in 'rbuf+len' bytes.                                 *
>418  *                                                        *
>419  *   Factored RCVDACK code out for general use.           *
>420  *                                                        *
>421  *   Changed REQUEST to return on ACK or NAK response.    *
>422  *                                                        *
>423  *   03/05/04:                                            *
>424  *                                                        *
>425  *   Changed code to avoid address modification (to allow *
>426  *   code to run in ROM).  This adds one cycle/byte.      *
>427  *                                                        *
>428  *   Changed RCVPKT digital PLL back to +/- 2 cycles/byte *
>429  *   because of page crossing variances in SENDPKT and    *
>430  *   RCVPKT LDA and STA ops that had been overlooked.     *
>431  *                                                        *
>432  *   09/04/03:                                            *
>433  *                                                        *
>434  *   Changed SENDPKT and RCVPKT to new bit timing by      *
>435  *   unrolling loops.                                     *
>436  *                                                        *
>437  *   Changed RCVPKT digital PLL to be +/- 1 cycle/byte    *
>438  *   instead of +/- 2 cycles/byte to tighten tolerances.  *
>439  *                                                        *
>440  **********************************************************
```

```
 51           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
```

```
 52           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
```

```
                 53           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
```

```
 54          put   NADABOOT2  ; Second-stage boot code
>1    ****************** Second-stage Boot *********************
>2
>3    *-----------------------------------------------------*
>4    *         Requester                  Master (ID=1)      *
>5    *  ======================     ======================  *
>6    *  GETID  REQ              ====>                       *
>7    *                          <==== GETID  ACK (ID)       *
>8    *  GETID  DACK  ($A5)      ====>                       *
>9    *-----------------------------------------------------*
```

```
                >11   ************************************************************
                >12   *                                                          *
                >13   *                        B O O T 2                         *
                >14   *                                                          *
                >15   *              Michael J. Mahon - May 13, 2004             *
                >16   *                   Adapted Apr 15, 2008                   *
                >17   *                   Revised Jan 24, 2009                   *
                >18   *                                                          *
                >19   *              Copyright (c) 2004, 2008, 2009              *
                >20   *                                                          *
                >21   *  Second boot stage for AppleCrate machines.              *
                >22   *                                                          *
                >23   *  The second boot stage gets control immediately after    *
                >24   *  the boot image has been loaded by the ROM boot code.    *
                >25   *                                                          *
                >26   *  The second stage performs the following steps:          *
                >27   *     1. Set "Inhibit GETID Out" to postpone GETID of      *
                >28   *        physically next board in AppleCrate.              *
                >29   *     2. Read Paddle 3 and use it to form a unique         *
                >30   *        "pseudo-ID" to minimize arbitration collisions.   *
                >31   *     3. Wait for "Inhibit GETID in" to go low.            *
                >32   *     4. Send a GETID request to the booting machine to    *
                >33   *        request a unique machine ID.  Then receive ACK    *
                >34   *        containing assigned machine ID.                   *
                >35   *     5. If any error, "beep" 100ms and retry step 3.      *
                >36   *     6. Respond to master with DACK using new ID.         *
                >37   *     7. Clear "Inhibit GETID Out" to enable GETID of      *
                >38   *        physically next board.                            *
                >39   *     8. Coldstart BASIC and set HIMEM to NadaNet base.    *
                >40   *     9. Set KSW to re-enter 'servelp' on input req.       *
                >41   *    10. Set warm start address (and PWREDUP byte) for     *
                >42   *        entry to NadaNet, so second-stage boot page       *
                >43   *        is free after boot.                               *
                >44   *    11. Jump to NadaNet entry address.                    *
                >45   *                                                          *
                >46   ************************************************************
                >47
B700: 4C 03 B7  >48   ep       jmp   BOOT2      ; Load page for boot loader
                >49
B703: 8D 5D C0  >50   BOOT2    sta   an2+1      ; Set Inhibit GETID out
B706: A9 03     >51            lda   #3         ; Set GETID 'request'
B708: 8D 4F B8  >52            sta   retrylim   ;  retrys to 3.
B70B: AD 8C 02  >53            lda   rbuf+frm-self+$280 ; Use ID of BOOTing
B70E: 8D 3F B8  >54            sta   sbuf+dst   ;  machine for GETID.
B711: A2 03     >55            ldx   #3         ; Set to read Paddle 3
B713: 20 1E FB  >56            jsr   PREAD      ; Read Paddle 3 (to Y)
B716: 98        >57            tya              ; Create pseudo-ID
B717: 09 80     >58            ora   #$80       ; (pseudo is >127)
B719: 20 C9 BA  >59            jsr   setid
B71C: 4C 22 B7  >60            jmp   :inhibit   ; No initial beep/delay.
                >61
B71F: 20 3A FF  >62   :retry   jsr   BELL       ; Delay 100ms & blink LED.
```

```
B722: AD 63 C0 >63   :inhibit lda   pb2         ; Test Inhibit GETID in
B725: 30 FB    >64            bmi   :inhibit    ;  and wait until low.
B727: A9 01    >65            lda   #reqto      ; Set RCVPKT timeout
B729: 8D 52 B8 >66            sta   tolim       ;  to min arb time.
B72C: A9 30    >67            lda   #r_GETID    ; Perform a GETID
B72E: 20 F3 BC >68            jsr   REQUEST     ;  request.
B731: B0 EC    >69            bcs   :retry      ; GETID failed.
B733: AD 49 B8 >70            lda   rbuf+adr    ; Save new assigned ID
B736: 8D CC 03 >71            sta   bootself    ;  in 'bootself',
B739: 20 C9 BA >72            jsr   setid       ;   etc.
B73C: A9 03    >73            lda   #rm_DACK    ; Send DACK.
B73E: 20 14 BE >74            jsr   SENDRSP
B741: 8D 5C C0 >75            sta   an2+0       ; Clear Inhibit GETID out.
B744: A9 32    >76            lda   #maxretry   ; Reset 'request'
B746: 8D 4F B8 >77            sta   retrylim    ;  retrys to default.
               >78            mov16 #:resume;KSW ; Set KSW to keep control
B749: A9 54    >78            lda   #:resume    ; Move 2 bytes
B74B: 85 38    >78            sta   KSW
B74D: A9 B7    >78            lda   #:resume/$100 ; high byte of immediate
B74F: 85 39    >78            sta   1+KSW
               >78            eom
B751: 4C 00 E0 >79            jmp   COLDSTRT    ; Coldstart Applesoft.
B754: A9 B8    >80   :resume  lda   #>entry     ; Set HIMEM
B756: 85 74    >81            sta   HIMEM+1     ;  and FRETOP to
B758: 85 70    >82            sta   FRETOP+1    ;   base of NadaNet.
B75A: A9 00    >83            lda   #0          ; Clear the
B75C: 85 D8    >84            sta   ONERR       ;  ONERR flag.
               >85            mov16 #warmstrt;KSW ; KSW goes to servelp.
B75E: A9 CD    >85            lda   #warmstrt   ; Move 2 bytes
B760: 85 38    >85            sta   KSW
B762: A9 03    >85            lda   #warmstrt/$100 ; high byte of immediate
B764: 85 39    >85            sta   1+KSW
               >85            eom
B766: A9 00    >86            lda   #<entry     ; Set SOFTEV vector to
B768: 8D F2 03 >87            sta   SOFTEV      ;  point to NadaNet entry.
B76B: A9 B8    >88            lda   #>entry
B76D: 8D F3 03 >89            sta   SOFTEV+1
B770: 49 A5    >90            eor   #$A5        ; Compute power-up byte
B772: 8D F4 03 >91            sta   PWREDUP     ;  and save it for RESET.
B775: 6C F2 03 >92            jmp   (SOFTEV)    ; Give NadaNet control...
               >93
               >94   ]end     align 256
B778: 00 00 00 >94            ds    *-1/256*256+256-*
               >94            eom
               >95   xboot    equ   *-]end      ; Slack space after boot2.
```

```
                 55           put   NADAVECTOR
                >1    ******************** Entry Points ************************
                >2
B800: 20 09 B9  >4    entry    jsr   INSTALL    ; BOOT entry: init and
B803: 20 E3 BA  >5    servelp  jsr   svrxkbd    ;  SERVE ignoring keypresses
B806: 4C 03 B8  >6             jmp   servelp    ;   forever...
                >7
B809: 4C 09 B9  >8    init     jmp   INSTALL    ; Initialize and return
B80C: 4C E6 BA  >9    serve    jmp   SERVER     ; Run request server
B80F: 4C 94 BB  >11   peek     jmp   PEEKREQ
B812: 4C 30 BC  >12   poke     jmp   POKEREQ
B815: 4C E2 BC  >13   call     jmp   CALLREQ
B818: 4C 78 BD  >14   putmsg   jmp   PUTMREQ
B81B: 4C 9A BD  >15   getmsg   jmp   GETMREQ
B81E: 4C 81 BB  >16   bcast    jmp   BCASTREQ
B821: 4C C8 BC  >17   bpoke    jmp   BPOKEREQ
B824: 4C E8 BB  >18   peekinc  jmp   PKINCREQ
B827: 4C EC BB  >19   peekpoke jmp   PKPOKREQ
B82A: 4C 28 BC  >20   run      jmp   RUNREQ
B82D: 4C 2C BC  >21   brun     jmp   BRUNREQ
B830: 4C 00 BF  >22   rcvctl   jmp   RCVCTL
B833: 4C 0A BF  >23   rcvptr   jmp   RCVPTR
B836: 4C 98 BF  >24   RARL=>AL jmp   rarl=>al
B839: 4C CF BF  >25   rcvlong  jmp   RCVLONG
                >42
                 56           put   NADAVARS
                >1    ********** Parameters and variables ************
                >2
                >6
B83C: 00        >7    self     db    0          ; Our own machine ID
B83D: 00 00 00  >8    sbuf     ds    lenctl     ; Control pkt send buffer
B845: 00 00 00  >9    rbuf     ds    lenctl     ; Control pkt receive buffer
B84D: 00 00     >10   locaddr  dw    0          ; Local address of req data
B84F: 32        >11   retrylim db    maxretry   ; Limit of REQUEST resends
B850: 00        >12   servecnt db    0          ; SERVE iterations (0=256)
                >13
                >14   parmsiz  equ   *-self     ; Size of parameter area
                >15
                >16   ************ Counters and Version **************
                >17
B851: 5C        >18   arbxv    db    arbx       ; Arbitrate X iters (modified)
B852: 01        >19   tolim    db    reqto      ; RCVPKT timeout limit
B853: 03        >20   reqctr   db    reqpidle   ; SERVER request counter
B854: 00        >21   reqretry db    0          ; xxxREQ retries remaining
B855: 00        >22   retrycnt db    0          ; REQUEST resend count
B856: 00 00     >23   errprot  dw    0          ; Protocol error count
B858: 00 00     >24   ckerr    dw    0          ; Checksum error count
B85A: 00 00     >25   frmcerr  dw    0          ; 'frmc' collision errors
B85C: 31        >26   nadaver  db    version    ; NadaNet version
                >27
                >37   idtable  equ   *          ; Address of bottom of vars
                >39
```

```
57           put   AMPERSAND
```

```
>2    ************************************************************
>3    *                                                          *
>4    *                    A M P E R N A D A                     *
>5    *                                                          *
>6    *            Michael J. Mahon - Oct 25, 2004               *
>7    *                  Revised May 3, 2010                     *
>8    *                                                          *
>9    *            Copyright (c) 2004, 2008, 2010                *
>10   *                                                          *
>11   *  Implements an ampersand (&) interface to NadaNet for    *
>12   *  Applesoft programs.  Reduces the need for PEEKs and     *
>13   *  POKEs to set up parameters, saving time and interface   *
>14   *  definitions.                                            *
>15   *                                                          *
>16   *  If an error occurs in a command execution routine,      *
>17   *  (signaled by Carry set upon return) the handler will,   *
>18   *  by default, throw a "DATA" (49) error, which will halt  *
>19   *  the program unless caught by an active ONERR.           *
>20   *                                                          *
>21   *  If an ampersand command is followed by a "#", then no   *
>22   *  execution error will be thrown, and the programmer      *
>23   *  is responsible for checking status by PEEKing 1 and 0.  *
>24   *                                                          *
>25   ************************************************************
>26
>27   ***************** Applesoft Definitions ******************
>28
>29   TXTPTR   equ   $B8        ; Current scan point
>30   VALTYP   equ   $11        ; $FF if var is STRING$
>31   INTFLG   equ   $12        ; $80 if var is INT%
>32   FORPNT   equ   $85        ; Ptr to var
>33   FAC      equ   $9D        ; Floating point accum
>34
>35   AMPVECT  equ   $3F5       ; JMP to ampersand handler
>36
>37   CHRGET   equ   $00B1      ; Get next text char
>38   CHRGOT   equ   $00B7      ; Get last text char
>39   ERROR    equ   $D412      ; Applesoft error handler
>40   SYNERR   equ   $DEC9      ; Syntax Error
>41   ADDON    equ   $D998      ; Advance TXTPTR by Y
>42   SYNCHR   equ   $DEC0      ; Current char must = A
>43   FRMNUM   equ   $DD67      ; Eval expr to FAC
>44   PTRGET   equ   $DFE3      ; Get var, ptr in (Y,A)
>45   GETBYT   equ   $E6F8      ; Eval expr to X
>46   GETADR   equ   $E752      ; Eval expr to (Y,A)
>47   FLO2     equ   $EBA0      ; Normalize FAC (C set)
>48   SETFOR   equ   $EB27      ; Pack FAC to (FORPNT)
>49
>50   ********************** Variables ************************
>51
>52   cmdptr   equ   $EC        ; Cmd table cursor
>53   cmdsave  equ   $ED        ; Current parm descriptor
```

```
                >54   disp     equ   $EF         ; Displacement to parm value
                >55
B85D: 00        >56   instald  db    0           ; Installed flag
B85E: 00        >57   nparms   db    0           ; # of parms seen
B85F: 00        >58   errstop  db    0           ; "Throw error" flag
B860: 00        >59   varcmd   db    0           ; var parm descriptor
B861: 00        >60   vartype  db    0           ; variable type
B862: 00 00     >61   varadr   da    0           ; variable address
```

```
                >63   ***************** Ampersand Command Table *****************
                >64
                >65   * Applesoft Token Definitions
                >66
                >67   CALL_t   equ   140
                >68   RUN_t    equ   172
                >69   POKE_t   equ   185
                >70   GET_t    equ   190
                >71   PEEK_t   equ   226
                >72
                >73   * Syntax string definitions
                >74
                >75   @        equ   self-1     ; NadaNet parameter origin
                >76   byte     equ   $00        ; Byte
                >77   word     equ   $40        ; Word
                >78   var      equ   $80        ; Numeric variable
                >79
                >80            err   parmsiz/63 ; Parm area < 64 bytes
                >81
                >82   iter     equ   servecnt-@.byte ; SERVER iteration count
                >83   dest     equ   sbuf+dst-@.byte ; Destination machine
                >84   addr     equ   sbuf+adr-@.word ; Address at destination
                >85   lngth    equ   sbuf+len-@.word ; Length
                >86   locadr   equ   locaddr-@.word  ; Local address
                >87   AX       equ   sbuf+len-@.word ; A,X regs for CALL
                >88   class    equ   sbuf+adr-@.word ; Class of message
                >89   incr     equ   sbuf+len-@.word ; Increment for PEEK INC
                >90   val      equ   sbuf+len-@.word ; Value for BPOKE, PEEKPOKE
                >91   n60ms    equ   retrylim-@.byte ; Request resend limit
                >92   lngth?   equ   rbuf+len-@.word.var ; Length (var)
                >93   val?     equ   rbuf+len-@.word.var ; Value (var)
                >94
                >95   * In command table, longer commands must precede shorter
                >96   * commands with a common prefix.
                >97
B864: 53 45 52  >98   cmdtable asc   'SERVE',00                ; &SERVE
B86A: 15 00     >99            db    iter,0
B86C: E6 BA     >100           da    SERVER
                >101
B86E: 50 55 54  >102           asc   'PUTMSG',00               ; &PUTMSG
B875: 04 46 48  >103           db    dest,class,lngth,locadr,0
B87A: 78 BD     >104           da    PUTMREQ
                >105
B87C: BE 4D 53  >106           db    GET_t,'M','S','G',0       ; &GETMSG
B881: 04 46 D0  >107           db    dest,class,lngth?,locadr,0
B886: 9A BD     >108           da    GETMREQ
                >109
B888: E2 49 4E  >110           db    PEEK_t,'I','N','C',0      ; &PEEKINC
B88D: 04 46 48  >111           db    dest,addr,incr,val?,0
B892: E8 BB     >112           da    PKINCREQ
                >113
B894: E2 B9 00  >114           db    PEEK_t,POKE_t,0           ; &PEEKPOKE
```

```
B897: 04 46 48 >115           db    dest,addr,val,val?,0
B89C: EC BB    >116           da    PKPOKREQ
               >117
B89E: E2 00    >118           db    PEEK_t,0                    ; &PEEK
B8A0: 04 46 48 >119           db    dest,addr,lngth,locadr,0
B8A5: 94 BB    >120           da    PEEKREQ
               >121
B8A7: B9 00    >122           db    POKE_t,0                    ; &POKE
B8A9: 04 46 48 >123           db    dest,addr,lngth,locadr,0
B8AE: 30 BC    >124           da    POKEREQ
               >125
B8B0: AC 00    >126           db    RUN_t,0                     ; &RUN
B8B2: 04 46 48 >127           db    dest,addr,lngth,locadr,0
B8B7: 28 BC    >128           da    RUNREQ
               >129
B8B9: 42 AC 00 >130           db    'B',RUN_t,0                 ; &BRUN
B8BC: 04 46 48 >131           db    dest,addr,lngth,locadr,0
B8C1: 2C BC    >132           da    BRUNREQ
               >133
B8C3: 8C 00    >134           db    CALL_t,0                    ; &CALL
B8C5: 04 46 48 >135           db    dest,addr,AX,0
B8C9: E2 BC    >136           da    CALLREQ
               >137
B8CB: 42 4F 4F >138           asc   'BOOT',00                   ; &BOOT
B8D0: 46 48 52 >139           db    addr,lngth,locadr,0
B8D4: 7D BB    >140           da    BOOTREQ
               >141
B8D6: 42 43 41 >142           asc   'BCAST',00                  ; &BCAST
B8DC: 46 48 52 >143           db    addr,lngth,locadr,0
B8E0: 81 BB    >144           da    BCASTREQ
               >145
B8E2: 42 B9 00 >146           db    'B',POKE_t,0                ; &BPOKE
B8E5: 46 48 00 >147           db    addr,val,0
B8E8: C8 BC    >148           da    BPOKEREQ
               >149
B8EA: 49 4E 49 >150           asc   'INIT',00                   ; &INIT
B8EF: 00       >151           db    0
B8F0: 97 BA    >152           da    INIT
               >153
B8F2: 54 49 4D >154           asc   'TIMEOUT',00                ; &TIMEOUT
B8FA: 14 00    >155           db    n60ms,0
B8FC: 52 BA    >156           da    timeout
               >157
B8FE: 49 44 54 >158           asc   'IDTBL',00                  ; &IDTBL
B904: D0 00    >159           db    val?,0
B906: 5E BA    >160           da    idtbl
               >161
B908: 00       >162           db    0          ; End of Command Table
```

```
                >164  ************************************************************
                >165  *                                                          *
                >166  *                     I N S T A L L                        *
                >167  *                                                          *
                >168  *           Michael J. Mahon - Oct 25, 2004                *
                >169  *                Revised Aug 16, 2008                      *
                >170  *                                                          *
                >171  *              Copyright (c) 2004, 2008                    *
                >172  *                                                          *
                >173  *  Installs AmperNada as first ampersand routine (if not   *
                >174  *  installed already) and chains to an existing routine.   *
                >175  *  if no routine is currently installed, it defaults to    *
                >176  *  "SYNTAX ERROR".                                         *
                >177  *                                                          *
                >178  ************************************************************
                >179
B909: AD 5D B8  >180  INSTALL  lda   instald    ; AmperNada installed?
B90C: D0 23     >181           bne   :exit      ; -Yes, don't repeat.
B90E: A9 4C     >182           lda   #$4C       ; -No, set flag and install.
B910: 8D 5D B8  >183           sta   instald
B913: CD F5 03  >184           cmp   AMPVECT    ; Is "&" vector a JMP?
B916: 8D F5 03  >185           sta   AMPVECT    ; (always set "jmp")
B919: D0 0C     >186           bne   :setvect   ; -No, just set vector.
                >187  :chain   mov16 AMPVECT+1;chain+1 ; -Yes, chain to it.
B91B: AD F6 03  >187           lda   AMPVECT+1  ; Move 2 bytes
B91E: 8D 4B B9  >187           sta   chain+1
B921: AD F7 03  >187           lda   1+AMPVECT+1
B924: 8D 4C B9  >187           sta   1+chain+1
                >187           eom
                >188  :setvect mov16 #AMPNADA;AMPVECT+1 ; set the vector.
B927: A9 34     >188           lda   #AMPNADA   ; Move 2 bytes
B929: 8D F6 03  >188           sta   AMPVECT+1
B92C: A9 B9     >188           lda   #AMPNADA/$100 ; high byte of immediate
B92E: 8D F7 03  >188           sta   1+AMPVECT+1
                >188           eom
B931: 4C 97 BA  >189  :exit    jmp   INIT       ; Initialize NadaNet.
```

```
                >191  ************************************************************
                >192  *                                                          *
                >193  *                    A M P E R N A D A                     *
                >194  *                                                          *
                >195  *            Michael J. Mahon - Oct 25, 2004               *
                >196  *                 Revised Nov 08, 2004                     *
                >197  *                                                          *
                >198  *                  Copyright (c) 2004                      *
                >199  *                                                          *
                >200  *  Implements an ampersand (&) interface to NadaNet for    *
                >201  *  Applesoft programs.  Reduces the need for PEEKs and     *
                >202  *  POKEs to set up parameters, saving time and interface   *
                >203  *  definitions.                                            *
                >204  *                                                          *
                >205  ************************************************************
                >206
B934: 08        >207  AMPNADA  php             ; Save status
B935: 48        >208          pha             ;  and A for chain.
B936: A2 00     >209          ldx   #0
B938: 8E 5E B8  >210          stx   nparms    ; # of parms supplied
B93B: 8E 60 B8  >211          stx   varcmd    ; Signal no var params seen
B93E: 8E 5F B8  >212          stx   errstop   ; Clear "throw err" flag.
B941: A0 00     >213  cmd      ldy   #0        ; Start compare at TXTPTR
B943: BD 64 B8  >214          lda   cmdtable,x ; Get command char
B946: D0 05     >215          bne   comp      ; -Not end, compare.
B948: 68        >216          pla             ; -End.  Restore A
B949: 28        >217          plp             ;   and status and chain
B94A: 4C C9 DE  >218  chain    jmp   SYNERR    ;    to next & handler.
                >219
B94D: D1 B8     >220  comp     cmp   (TXTPTR),y ; Does cmd match text?
B94F: D0 09     >221          bne   :skipcmd  ; -No, skip this one.
B951: C8        >222          iny             ; -Yes, advance.
B952: E8        >223          inx
B953: BD 64 B8  >224          lda   cmdtable,x ; End of command?
B956: D0 F5     >225          bne   comp      ; -No, keep comparing.
B958: F0 11     >226          beq   :doit     ; -Yes, go do it.
                >227
B95A: E8        >228  :skipcmd inx             ; Skip to end of
B95B: BD 64 B8  >229          lda   cmdtable,x ;  current cmd string
B95E: D0 FA     >230          bne   :skipcmd
B960: E8        >231  :skipp   inx             ; Skip to end of
B961: BD 64 B8  >232          lda   cmdtable,x ;  current parm vect
B964: D0 FA     >233          bne   :skipp
B966: E8        >234          inx             ; Pass end mark
B967: E8        >235          inx             ;  and action
B968: E8        >236          inx             ;   routine address.
B969: D0 D6     >237          bne   cmd       ; Go check next command.
                >238
B96B: 68        >239  :doit    pla             ; Discard entry A
B96C: 68        >240          pla             ;  and status.
B96D: B1 B8     >241          lda   (TXTPTR),y ; Look at next character.
B96F: C8        >242          iny             ; (provisional match)
```

```
B970: C9 23    >243           cmp   #'#'        ; Is it "#"?
B972: F0 04    >244           beq   :advance    ; -Yes, don't throw error.
B974: 88       >245           dey               ; -No, don't match, and
B975: EE 5F B8 >246           inc   errstop     ;   set throw err flag.
B978: 20 98 D9 >247  :advance jsr   ADDON        ; Advance TXTPTR past cmd
B97B: A9 28    >248           lda   #'('        ; Require initial "("
B97D: 20 C0 DE >249  :nxparm  jsr   SYNCHR      ; Syntax err if no match.
B980: F0 61    >250           beq   :synerr     ; End not expected.
B982: 86 EC    >251           stx   cmdptr      ; Save for :done case
B984: C9 29    >252           cmp   #')'        ; Found a ")"?
B986: F0 5E    >253           beq   :done       ; -Yes, end of parm list.
B988: EE 5E B8 >254           inc   nparms      ; -No, another parm.
B98B: E8       >255           inx               ; Advance ptr and
B98C: BD 64 B8 >256           lda   cmdtable,x  ;  get parm descriptor.
B98F: F0 52    >257           beq   :synerr     ; Too many parms.
B991: 85 ED    >258           sta   cmdsave     ; Save descriptor
B993: 29 3F    >259           and   #$3F        ; Mask displacement
B995: 85 EF    >260           sta   disp        ;  and save it.
B997: 86 EC    >261           stx   cmdptr      ; Save pointer.
B999: 24 ED    >262           bit   cmdsave     ; Test parm type.
B99B: 30 20    >263           bmi   :var        ; -Var parm
B99D: 50 12    >264           bvc   :byte       ; -Byte value parm
B99F: 20 67 DD >265           jsr   FRMNUM      ; -Word value parm
B9A2: 20 52 E7 >266           jsr   GETADR      ; Word val to Y,A
B9A5: A6 EF    >267           ldx   disp
B9A7: 9D 3C B8 >268           sta   @+1,x       ; Store the value
B9AA: 98       >269           tya
B9AB: 9D 3B B8 >270           sta   @,x
B9AE: 4C D4 B9 >271           jmp   :more?
               >272
B9B1: 20 F8 E6 >273  :byte    jsr   GETBYT      ; Byte value to X
B9B4: A4 EF    >274           ldy   disp
B9B6: 8A       >275           txa
B9B7: 99 3B B8 >276           sta   @,y         ; Store the value
B9BA: 4C D4 B9 >277           jmp   :more?
               >278
B9BD: A5 ED    >279  :var     lda   cmdsave     ; Save the parm
B9BF: 8D 60 B8 >280           sta   varcmd      ;  descriptor.
B9C2: 20 E3 DF >281           jsr   PTRGET      ; Get var ptr in (A,Y)
B9C5: 8D 62 B8 >282           sta   varadr      ;  and save var
B9C8: 8C 63 B8 >283           sty   varadr+1    ;   address.
B9CB: A5 11    >284           lda   VALTYP      ; $FF if string
B9CD: D0 14    >285           bne   :synerr     ; String not allowed.
B9CF: A5 12    >286           lda   INTFLG      ; $80 if INT%
B9D1: 8D 61 B8 >287           sta   vartype     ; Save for later use
B9D4: 20 B7 00 >288  :more?   jsr   CHRGOT      ; Check current test char.
B9D7: F0 0A    >289           beq   :synerr     ; End not expected.
B9D9: C9 29    >290           cmp   #')'        ; Closing ")"?
B9DB: F0 09    >291           beq   :done       ; -Yes, finish.
B9DD: A6 EC    >292           ldx   cmdptr      ; -No, more parms.
B9DF: A9 2C    >293           lda   #','        ; Require a comma.
B9E1: D0 9A    >294           bne   :nxparm     ; (always)
```

```
                >295
B9E3: 4C C9 DE >296  :synerr  jmp   SYNERR     ; SYNTAX ERROR
                >297
B9E6: 20 B1 00 >298  :done    jsr   CHRGET     ; Pass the ")"
B9E9: A6 EC    >299           ldx   cmdptr
B9EB: E8       >300  :skipit  inx              ; Skip to end
B9EC: BD 64 B8 >301           lda   cmdtable,x ;  of parm descriptors.
B9EF: D0 FA    >302           bne   :skipit
                >303           mov16 cmdtable+1,x;$00 ; Action routine
B9F1: BD 65 B8 >303           lda   cmdtable+1,x ; Move 2 bytes
B9F4: 85 00    >303           sta   $00
B9F6: BD 66 B8 >303           lda   1+cmdtable+1,x
B9F9: 85 01    >303           sta   1+$00
                >303           eom
B9FB: 20 11 BA >304           jsr   :jmp       ; Call the action routine
B9FE: 85 00    >305           sta   $00         ; Save returned A
BA00: A9 00    >306           lda   #0
BA02: 2A       >307           rol               ; C to low bit
BA03: 85 01    >308           sta   $01         ; Save returned Carry
BA05: F0 0D    >309           beq   :noerr      ; No error, continue.
BA07: AD 5F B8 >310           lda   errstop     ; Throw error?
BA0A: F0 0D    >311           beq   :rts        ; -No, just return.
BA0C: A2 31    >312           ldx   #49         ; -Yes, throw "DATA"
BA0E: 4C 12 D4 >313           jmp   ERROR       ;   error.
                >314
BA11: 6C 00 00 >315  :jmp     jmp   ($00)       ; To action routine.
                >316
BA14: AD 60 B8 >317  :noerr   lda   varcmd      ; Var parm passed?
BA17: D0 01    >318           bne   :store      ; -Yes, store into it.
BA19: 60       >319  :rts     rts               ; -No, return.
                >320
BA1A: 29 3F    >321  :store   and   #$3F        ; Mask displacement
BA1C: A8       >322           tay
BA1D: B9 3B B8 >323           lda   @,y         ; Get low byte
BA20: AA       >324           tax               ; X = lo byte of value
BA21: A9 00    >325           lda   #0          ; Hi byte if byte value
BA23: 2C 60 B8 >326           bit   varcmd      ; Is it byte or word?
BA26: 50 03    >327           bvc   :byteval    ; -Byte, use 0 hi byte
BA28: B9 3C B8 >328           lda   @+1,y       ; -Word, get hi byte
BA2B: A8       >329  :byteval tay               ; Y = hi byte of value
                >330           mov16 varadr;FORPNT ; Address of variable
BA2C: AD 62 B8 >330           lda   varadr      ; Move 2 bytes
BA2F: 85 85    >330           sta   FORPNT
BA31: AD 63 B8 >330           lda   1+varadr
BA34: 85 86    >330           sta   1+FORPNT
                >330           eom
BA36: AD 61 B8 >331           lda   vartype     ; INT% or FLOAT variable?
BA39: 10 0A    >332           bpl   :float      ; -FLOAT
BA3B: 98       >333           tya               ; -INT%
BA3C: A0 00    >334           ldy   #0          ; Store hi byte
BA3E: 91 85    >335           sta   (FORPNT),y ;  in INT% variable.
BA40: C8       >336           iny               ; Point to lo byte
```

```
BA41: 8A       >337           txa              ; Store lo byte
BA42: 91 85    >338           sta   (FORPNT),y ;  in INT% variable.
BA44: 60       >339           rts
               >340
BA45: 84 9E    >341  :float   sty   FAC+1       ; Hi byte to FAC
BA47: 86 9F    >342           stx   FAC+2       ; Lo byte to FAC
BA49: A2 90    >343           ldx   #$90        ; Binary point 16 bits right
BA4B: 38       >344           sec              ; (Don't negate FAC)
BA4C: 20 A0 EB >345           jsr   FLO2        ; Normalize FAC
BA4F: 4C 27 EB >346           jmp   SETFOR      ; Pack FAC into variable.
```

```
                >348  ************************************************************
                >349  *                                                          *
                >350  *                  &TIMEOUT ([n60ms])                      *
                >351  *                                                          *
                >352  *           Michael J. Mahon - Oct 28, 2004                *
                >353  *                 Revised May 3, 2010                      *
                >354  *                                                          *
                >355  *              Copyright (c) 2004, 2010                    *
                >356  *                                                          *
                >357  *  Set new request timeout value in units of 60 ms.       *
                >358  *                                                          *
                >359  *  If no value is supplied, reset timeout to default.     *
                >360  *                                                          *
                >361  ************************************************************
                >362
BA52: AD 5E B8  >363  timeout  lda   nparms     ; Parm supplied?
BA55: D0 05     >364           bne   null       ; -Yes, timeout set.
BA57: A9 32     >365           lda   #maxretry  ; -No, restore
BA59: 8D 4F B8  >366           sta   retrylim   ;   the default.
BA5C: 18        >367  null     clc              ; No error
BA5D: 60        >368           rts
```

```
                >370  ************************************************************
                >371  *                                                          *
                >372  *                     &IDTBL (val?)                        *
                >373  *                                                          *
                >374  *           Michael J. Mahon - Nov 05, 2004                *
                >375  *                                                          *
                >376  *                 Copyright (c) 2004                       *
                >377  *                                                          *
                >378  *  Return address of 'idtable' in parm variable.           *
                >379  *                                                          *
                >380  ************************************************************
                >381
                >382  idtbl    mov16 #idtable;rbuf+len ; Put addr in rbuf
BA5E: A9 5D     >382            lda   #idtable   ; Move 2 bytes
BA60: 8D 4B B8  >382            sta   rbuf+len
BA63: A9 B8     >382            lda   #idtable/$100 ; high byte of immediate
BA65: 8D 4C B8  >382            sta   1+rbuf+len
                >382            eom
BA68: 18        >383            clc
BA69: 60        >384            rts
```

```
                 58           put   INITSERVE
                >2    *** Table of service routines used by SERVER ***
                >3
BA6A: C5 BB     >4    service  dw    PEEKSRV    ; Table of service routines
BA6C: 6D BC     >5             dw    POKESRV    ; (Must be in order)
BA6E: E7 BC     >6             dw    CALLSRV
BA70: 51 BB     >11            dw    ]PROTERR   ; (Error if PUTMSG)
BA72: 51 BB     >12            dw    ]PROTERR   ; (Error if GETMSG)
BA74: 51 BB     >17            dw    ]PROTERR   ; (Error if GETID)
BA76: 51 BB     >19            dw    ]PROTERR   ; (Error if non-bcast BOOT)
BA78: 98 BF     >20            dw    rarl=>al   ; (BCAST data up to caller)
BA7A: D3 BC     >21            dw    BPOKESRV
BA7C: 05 BC     >22            dw    PKINCSRV
BA7E: 09 BC     >23            dw    PKPOKSRV
BA80: 58 BC     >27            dw    RUNSRV
BA82: 6D BC     >29            dw    BRUNSRV
                >30
                >31   * Version message printed by INIT
                >32
BA84: CE C1 C4  >33   vermsg   asc   "NADANET "
BA8C: B3        >34            db    version/16."0" ; Major version #
BA8D: AE        >35            asc   "."
BA8E: B1        >36            db    version&$0F."0" ; Minor version #
BA8F: AC A0 C9  >40            asc   ", ID = $"
                >41   verlen   equ   *-vermsg   ; Length of msg
```

```
                >43   ************************************************************
                >44   *                                                          *
                >45   *                         I N I T                          *
                >46   *                                                          *
                >47   *              Michael J. Mahon - Mar 5, 2004              *
                >48   *                   Revised May 21, 2008                   *
                >49   *                                                          *
                >50   *           Copyright (c) 1996, 2004, 2005, 2008           *
                >51   *                                                          *
                >52   *  Initialize NADANET, sign on, and return to caller.      *
                >53   *                                                          *
                >54   ************************************************************
                >55
BA97: AD CC 03  >56   INIT     lda   bootself   ; Set up ID from BOOT
BA9A: 20 C9 BA  >57            jsr   setid
BA9D: B0 29     >58            bcs   :err       ; Bad ID, no INIT.
BA9F: A9 4C     >59            lda   #$4C       ; Set warmstrt JMP to
BAA1: 8D CD 03  >60            sta   warmstrt   ;  servlp (& nadapage)
                >61            mov16 #servelp;warmstrt+1
BAA4: A9 03     >61            lda   #servelp   ; Move 2 bytes
BAA6: 8D CE 03  >61            sta   warmstrt+1
BAA9: A9 B8     >61            lda   #servelp/$100 ; high byte of immediate
BAAB: 8D CF 03  >61            sta   1+warmstrt+1
                >61            eom
BAAE: 20 8B FD  >62            jsr   CROUT1     ; New line.
BAB1: A0 00     >63            ldy   #0
BAB3: B9 84 BA  >64   :msgloop lda   vermsg,y   ; Print version message
BAB6: 20 ED FD  >65            jsr   COUT
BAB9: C8        >66            iny
BABA: C0 13     >67            cpy   #verlen
BABC: 90 F5     >68            bcc   :msgloop
BABE: AD 3C B8  >69            lda   self       ;  and current ID.
BAC1: 20 DA FD  >70            jsr   PRBYTE     ;   (in hex)
BAC4: 20 8B FD  >71            jsr   CROUT1     ; New line.
BAC7: 18        >72            clc              ; Good return.
BAC8: 60        >73   :err     rts
```

```
                >76   ************************************************************
                >77   *                                                          *
                >78   *                        S E T I D                         *
                >79   *                                                          *
                >80   *              Michael J. Mahon - May 13, 2004             *
                >81   *                    Revised Aug 17, 2008                  *
                >82   *                                                          *
                >83   *                  Copyright (c) 2004, 2008                *
                >84   *                                                          *
                >85   *  Set machine ID to contents of A register and reset      *
                >86   *  the arbitration delay to 'arbtime' plus 22 cycles       *
                >87   *  times the machine ID, to avoid collisions.              *
                >88   *                                                          *
                >89   *  Delay from last arbitration poll to bus lock is 10      *
                >90   *  cycles, so 22 (2 * 11 cycles) increment provides a      *
                >91   *  little insurance.                                       *
                >92   *                                                          *
                >93   ************************************************************
                >94
BAC9: 8D 3C B8  >95   setid    sta   self       ; Machine ID
BACC: 8D 40 B8  >96            sta   sbuf+frm   ; Set sender field.
BACF: 49 FF     >97            eor   #$FF       ; Complement ID
BAD1: 8D 3E B8  >98            sta   sbuf+frmc  ;  for collision detect.
BAD4: 49 FF     >99            eor   #$FF       ; Back to ID
BAD6: 38        >100           sec              ; Anticipate error.
BAD7: F0 09     >101           beq   :err       ; -Error if zero.
BAD9: 18        >102           clc              ; Anticipate no error.
BADA: 30 03     >103           bmi   :setarb    ; -Use temp ID (>127)
BADC: 0A        >104           asl              ; Mult ID by 2
BADD: 69 5C     >105           adc   #arbx      ;  and add to base
BADF: 8D 51 B8  >106  :setarb  sta   arbxv      ;   arb delay.
BAE2: 60        >107  :err     rts
```

```
                >110  ************************************************************
                >111  *                                                          *
                >112  *                        S E R V E R                       *
                >113  *                                                          *
                >114  *              Michael J. Mahon - May 5, 1996              *
                >115  *                   Revised Oct 06, 2008                   *
                >116  *                                                          *
                >117  *              Copyright (c) 1996, 2004, 2008              *
                >118  *                                                          *
                >119  *  SERVER continually listens to the net, receiving all    *
                >120  *  packets, and responding to control packets directed      *
                >121  *  to 'self'.  If a key is pressed or a request handled,    *
                >122  *  SERVER returns. C = 0 if count expired and is set as     *
                >123  *  the request server left it if a request was handled.     *
                >124  *                                                          *
                >125  *  To minimize missed polls, SERVER temporarily raises     *
                >126  *  RCVPKT's timeout to 20ms. from the normal value equal   *
                >127  *  to the minimum arbitration time.                        *
                >128  *                                                          *
                >129  *  For every request code, there is a corresponding        *
                >130  *  server routine.  SERVER invokes these routines to       *
                >131  *  satisfy the service requests it receives.  Upon entry   *
                >132  *  to 'xxxSRV', C = 0 and (X) = (rbuf+rqmd).               *
                >133  *                                                          *
                >134  *  To ensure that the next packet received is the start    *
                >135  *  packet of a request protocol, it is necessary to wait   *
                >136  *  for the net to be idle or locked for at least the min   *
                >137  *  arbitration time before receiving a request.  (Note     *
                >138  *  that broadcast requests begin with the network in a     *
                >139  *  locked state.)                                           *
                >140  *                                                          *
                >141  *  The entry point 'svrxkbd' is provided for 'servelp',    *
                >142  *  which ignores keyboard input.                           *
                >143  *                                                          *
                >144  ************************************************************
                >145
BAE3: AD 10 C0  >146  svrxkbd  lda   kbstrobe   ; Ignore any keypress
                >147
BAE6: A9 08     >148  SERVER   lda   #idleto    ; While polling, raise
BAE8: 8D 52 B8  >149           sta   tolim      ;  RCVPKT timeout to 20ms.
BAEB: A2 3A     >150  :resync  ldx   #servegap  ; Delay min arb time
BAED: CD E8 C0  >151           cmp   zipslow    ; Zip Chip to 1MHz mode.
BAF0: AC 62 C0  >152           ldy   drecv      ; Sample net state.
BAF3: 98        >153  :waitidl tya
BAF4: 4D 62 C0  >154           eor   drecv      ; Has net changed?
BAF7: 30 F2     >155           bmi   :resync    ; -Yes, restart timing.
BAF9: CA        >156           dex              ; -No, count it down.
BAFA: D0 F7     >157           bne   :waitidl   ; -Keep waiting.
BAFC: AD 00 C0  >158  :serve   lda   keybd      ; Check if key pressed.
BAFF: 30 75     >159           bmi   :exit      ; -Yes, return.
BB01: 20 00 BF  >160           jsr   RCVCTL     ; Receive ctl pkt to 'rbuf'
BB04: B0 69     >161           bcs   :err       ; -Timeout or Cksum err.
```

```
BB06: AD 46 B8 >162           lda   rbuf+frmc  ; -Cksum OK, verify that
BB09: 49 FF    >163           eor   #$FF       ;   complement of 'frmc'
BB0B: CD 48 B8 >164           cmp   rbuf+frm   ;    is equal to 'frm'.
BB0E: D0 55    >165           bne   :frmcerr   ; -No, count collisions.
BB10: AD 47 B8 >166           lda   rbuf+dst   ; -Yes, good packet.
BB13: F0 2D    >167           beq   :bcastck   ; Broadcast packet OK?
BB15: CD 3C B8 >168           cmp   self       ; Directed to us?
BB18: D0 3A    >169           bne   :skip      ; -No, just keep time.
BB1A: AD 45 B8 >170  :bcast   lda   rbuf+rqmd  ; -Yes, get 'rqmd'
BB1D: AA       >171           tax              ;   and save in X.
BB1E: 29 07    >172           and   #modmask   ; Is the modifier
BB20: C9 01    >173           cmp   #rm_REQ    ;  a Request?
BB22: D0 2D    >174           bne   ]PROTERR   ; -No, protocol error.
BB24: 8A       >175           txa              ; -Yes, check request.
BB25: 29 F8    >176           and   #reqmask
BB27: F0 28    >177           beq   ]PROTERR   ; Code must be > 0
BB29: C9 70    >178           cmp   #maxreq    ;  and < maxreq.
BB2B: B0 24    >179           bcs   ]PROTERR   ; Invalid request.
BB2D: 4A       >180           lsr              ; Req code is * 8,
BB2E: 4A       >181           lsr              ;  so divide by 4. (C=0)
BB2F: A8       >182           tay              ; Index of service routine
               >183           mov16 service-2,y;address ; Set up address
BB30: B9 68 BA >183           lda   service-2,y ; Move 2 bytes
BB33: 85 FC    >183           sta   address
BB35: B9 69 BA >183           lda   1+service-2,y
BB38: 85 FD    >183           sta   1+address
               >183           eom
BB3A: A9 01    >184           lda   #reqto     ; Reset timeout to min
BB3C: 8D 52 B8 >185           sta   tolim      ;  arbitration time.
BB3F: 6C FC 00 >186           jmp   (address)  ; Jump to service routine.
               >187
BB42: AD 45 B8 >188  :bcastck lda   rbuf+rqmd  ; Ck broadcast valid..
BB45: C9 49    >189           cmp   #r_BPOKE+rm_REQ ; BPOKE request?
BB47: F0 D1    >190           beq   :bcast     ; -Yes, process request.
BB49: C9 41    >191           cmp   #r_BCAST+rm_REQ ; Broadcast BCAST req?
BB4B: F0 CD    >192           beq   :bcast     ; -Yes.
BB4D: C9 39    >193           cmp   #r_BOOT+rm_REQ ; Broadcast BOOT req?
BB4F: F0 03    >194           beq   :skip      ; -Yes, ignore.
BB51: 20 8F BF >195  ]PROTERR jsr   PROTERR    ; Record protocol error
BB54: CE 53 B8 >196  :skip    dec   reqctr     ; Enough requests seen?
BB57: D0 92    >197           bne   :resync    ; -No, re-sync SERVER.
BB59: A9 03    >198           lda   #reqpidle  ; -Yes, about 20ms used.
BB5B: 8D 53 B8 >199           sta   reqctr     ; Reset counter.
BB5E: CE 50 B8 >200           dec   servecnt   ; Enough iterations?
BB61: F0 13    >201           beq   :exit      ; -Yes, return.
BB63: D0 86    >202           bne   :resync    ; -No, re-sync SERVER.
               >203
               >204  :frmcerr inc16 frmcerr    ; Count sync'd collisions.
BB65: EE 5A B8 >204           inc   frmcerr    ; Increment 16-bit word.
BB68: D0 03    >204           bne   *+5        ; - No carry.
BB6A: EE 5B B8 >204           inc   frmcerr+1  ; Propagate carry.
               >204           eom
```

```
BB6D: D0 E5    >205           bne   :skip      ; (always)
               >206
BB6F: D0 E3    >207  :err     bne   :skip      ; -Cksum error.
BB71: CE 50 B8 >208           dec   servecnt   ; -Timeout.  Enough?
BB74: D0 86    >209           bne   :serve     ; -No, keep serving.
BB76: A9 01    >210  :exit    lda   #reqto     ; -Yes, restore normal
BB78: 8D 52 B8 >211           sta   tolim      ;   request timeout,
BB7B: 18       >212           clc              ;    clear Carry
BB7C: 60       >213           rts              ;     and return.
```

```
>217  *-------------------------------------------------------*
>218  *            Broadcast Boot & Bcast Protocol            *
>219  *-------------------------------------------------------*
>220  *          Master                      Slaves           *
>221  *  ======================     ======================  *
>222  *  Bxxx   REQ (addr,leng)  ====>                       *
>223  *                   (800 cyc. delay)                    *
>224  *                     Data  ====>                       *
>225  *                           :                           *
>226  *                     Data  ====>                       *
>227  *-------------------------------------------------------*
```

```
                >229  ************************************************************
                >230  *                                                          *
                >231  *          B O O T R E Q   &   B C A S T R E Q             *
                >232  *                                                          *
                >233  *            Michael J. Mahon - May 14, 2004               *
                >234  *                  Revised Apr 28, 2010                    *
                >235  *                                                          *
                >236  *             Copyright (c) 2004, 2008, 2010               *
                >237  *                                                          *
                >238  *  Broadcast request for all waiting machines to receive   *
                >239  *  data of 'sbuf+len' length.                              *
                >240  *                                                          *
                >241  *  BOOTREQ is handled by all machines awaiting boot.       *
                >242  *  The boot image following is loaded at 'sbuf+adr' and    *
                >243  *  control is passed to the boot image.                    *
                >244  *                                                          *
                >245  *  BCASTREQ is handled by all machines awaiting BCAST      *
                >246  *  data.  'sbuf+adr' is the "tag" for the data following,  *
                >247  *  that is ignored or received by waiting machines based   *
                >248  *  on their state and the tag value.                       *
                >249  *                                                          *
                >250  *  Since these requests are broadcast, they do not get     *
                >251  *  ACKs from their destination(s), but simply send their   *
                >252  *  data blindly.  If errors occur, waiting machines will   *
                >253  *  continue to wait for good data, so verification of      *
                >254  *  proper operation must be handled separately.            *
                >255  *                                                          *
                >256  *  Because broadcast data is sent "open loop", and since   *
                >257  *  BCAST clients may require time to determine whether     *
                >258  *  and how they should receive the following data, these   *
                >259  *  protocols delay for 800 cycles between the request      *
                >260  *  and the sending of data.                                 *
                >261  *                                                          *
                >262  *  BOOTREQ & BCASTREQ do the following steps:               *
                >263  *     1. Sets up the request                               *
                >264  *     2. Does a broadcast arbitration to seize the net     *
                >265  *        and delay 20ms to resolve any collisions and      *
                >266  *        allow slow pollers to reach their RCVPKT holds    *
                >267  *     3. Sends the request, with address/tag and length    *
                >268  *     4. Waits 800 cyc. for clients to prepare to          *
                >269  *        receive the data (or not).                        *
                >270  *     5. Sends the boot code/data stream                   *
                >271  *                                                          *
                >272  ************************************************************
                >273
BB7D: A9 39     >274  BOOTREQ  lda   #r_BOOT+rm_REQ
BB7F: D0 02     >275           bne   ]doit      ; (always)
                >276
BB81: A9 41     >277  BCASTREQ lda   #r_BCAST+rm_REQ ; BCAST request
BB83: 8D 3D B8  >278  ]doit    sta   sbuf+rqmd
BB86: 20 CB BD  >279           jsr   BCASTARB   ; Bcast arbitrate & lock bus
BB89: 20 26 BE  >280           jsr   SENDCTL    ; Send the BOOT request.
```

```
BB8C: 20 E9 BE >281          jsr   lasl=>al   ; Local start address & length
BB8F: A2 A0    >282          ldx   #800/5     ; Delay 800 cycles,
BB91: 4C AD BF >283          jmp   DSENDLNG   ;  send data and return.
```

```
 59           put   PEEKPOKECALL
>2    *-------------------------------------------------------*
>3    *          Requester                    Server          *
>4    *  ======================     ======================  *
>5    *  PEEK   REQ (addr,leng)  ====>                       *
>6    *                          <==== PEEK   ACK            *
>7    *                          <==== Data (if >4 bytes)    *
>8    *                            :                         *
>9    *                          <==== Data                  *
>10   *-------------------------------------------------------*
```

```
                >14   ************************************************************
                >15   *                                                          *
                >16   *                      P E E K R E Q                       *
                >17   *                                                          *
                >18   *              Michael J. Mahon - May 5, 1996              *
                >19   *                  Revised May 21, 2008                    *
                >20   *                                                          *
                >21   *                Copyright (c) 1996, 2008                  *
                >22   *                                                          *
                >23   *  Request machine 'sbuf+dst' to send 'sbuf+len' bytes     *
                >24   *  at its 'sbuf+adr', and put them at location 'locaddr'.  *
                >25   *                                                          *
                >26   *  PEEKREQ, like other requests, will retry the request    *
                >27   *  in case of error, up to 'maxreqrt' times.  If errors    *
                >28   *  persist, it will return with Carry set.                 *
                >29   *                                                          *
                >30   *  PEEKREQ does the following steps:                       *
                >31   *     1. Make the PEEK request (and receive the ACK)       *
                >32   *     2. Receive 'sbuf+len' bytes of data into 'locaddr'   *
                >33   *     3. Retry in case of error up to 'maxreqrt' times     *
                >34   *                                                          *
                >35   ************************************************************
                >36
BB94: A9 03     >37   PEEKREQ  lda   #maxreqrt  ; Set request retry
BB96: 8D 54 B8  >38            sta   reqretry   ;  counter.
BB99: A9 08     >39   :retry   lda   #r_PEEK    ; Send PEEK request.
BB9B: 20 F3 BC  >40            jsr   REQUEST
BB9E: B0 1E     >41            bcs   :failed
BBA0: 20 E9 BE  >42            jsr   lasl=>al   ; Set up address/length
BBA3: A5 FF     >43            lda   length+1   ; If length
BBA5: D0 12     >44            bne   :long      ;  is >255 bytes, or
BBA7: A4 FE     >45            ldy   length     ;    if length is
BBA9: F0 19     >46            beq   :done      ; (length = 0!)
BBAB: C0 05     >47            cpy   #5         ;      > 4 bytes,
BBAD: B0 0A     >48            bcs   :long      ;   receive multiple pkts.
BBAF: 88        >49            dey              ; Move short response
BBB0: B9 49 B8  >50   :short   lda   rbuf+adr,y ;  to local data address.
BBB3: 91 FC     >51            sta   (address),y
BBB5: 88        >52            dey
BBB6: 10 F8     >53            bpl   :short
BBB8: 60        >54            rts              ; ...and return.
                >55
BBB9: 20 CF BF  >56   :long    jsr   RCVLONG    ; Receive multiple packets
BBBC: 90 06     >57            bcc   :done      ; No problem.
BBBE: CE 54 B8  >58   :failed  dec   reqretry   ; Dec request retry count
BBC1: D0 D6     >59            bne   :retry     ; Try until OK or exhausted,
BBC3: 38        >60            sec              ;  then return with C set.
BBC4: 60        >61   :done    rts
```

```
                >65   ************************************************************
                >66   *                                                          *
                >67   *                      P E E K S R V                       *
                >68   *                                                          *
                >69   *              Michael J. Mahon - May 5, 1996              *
                >70   *                   Revised May 21, 2008                   *
                >71   *                                                          *
                >72   *              Copyright (c) 1996, 2005, 2008              *
                >73   *                                                          *
                >74   *  Service machine 'rbuf+frm's request to send 'rbuf+len'* 
                >75   *  bytes of data from our 'rbuf+adr'.                       *
                >76   *                                                          *
                >77   *  PEEKSRV does the following steps:                       *
                >78   *     1. Check 'rbuf+len' for a 1..4 byte request          *
                >79   *     2. Send the ACK packet (with data, if short)         *
                >80   *     3. If long, send multiple response packets           *
                >81   *                                                          *
                >82   ************************************************************
                >83
BBC5: 20 98 BF  >84   PEEKSRV  jsr   rarl=>al   ; Set address/length.
BBC8: A5 FF     >85            lda   length+1   ; Check for long response
BBCA: D0 14     >86            bne   :long
BBCC: A4 FE     >87            ldy   length     ; Check for < 5 bytes.
BBCE: F0 0D     >88            beq   :nullreq   ; length = 0.
BBD0: C0 05     >89            cpy   #5
BBD2: B0 0C     >90            bcs   :long      ; - No, longer.
BBD4: 88        >91            dey              ; - Yes, move response
BBD5: B1 FC     >92   :short   lda   (address),y ;   data into ACK packet.
BBD7: 99 41 B8  >93            sta   sbuf+adr,y
BBDA: 88        >94            dey
BBDB: 10 F8     >95            bpl   :short
BBDD: 4C 12 BE  >96   :nullreq jmp   SENDACK    ; Send ACK with response.
                >97
BBE0: 20 12 BE  >98   :long    jsr   SENDACK    ; ACK the request.
BBE3: A2 1C     >99            ldx   #140/5     ; Allow requester to receive.
BBE5: 4C AD BF >100            jmp   DSENDLNG   ; Send long response.
```

```
                >102  *-------------------------------------------------------*
                >103  *         Requester                    Server           *
                >104  *  ======================     ======================    *
                >105  *  PEEKINC REQ (addr,inc)  ====>                        *
                >106  *                          <==== PEEKINC ACK (oldval)   *
                >107  *-------------------------------------------------------*
                >108
                >111  *********************************************************
                >112  *                                                       *
                >113  *          P K I N C R E Q ,   P K P O K R E Q          *
                >114  *                                                       *
                >115  *            Michael J. Mahon - Nov 05, 2004            *
                >116  *                                                       *
                >117  *               Copyright (c) 2004, 2010                *
                >118  *                                                       *
                >119  *  Request machine 'sbuf+dst' to return 2 bytes at its  *
                >120  *  'sbuf+adr', then increment that value by 'sbuf+len'  *
                >121  *  (PEEKINC), or set the value to 'sbuf+len' (PEEKPOKE). *
                >122  *  The returned, unchanged value is at 'rbuf+len'.      *
                >123  *                                                       *
                >124  *  These requests, like others, will retry the request  *
                >125  *  in case of error, up to 'maxreqrt' times.  If errors *
                >126  *  persist, SIMPLREQ will return with Carry set.        *
                >127  *                                                       *
                >128  *  PEEKINC and PEEKPOKE do the following steps:         *
                >129  *     1. Make the request (and receive the ACK)         *
                >130  *     2. Retry in case of error up to 'maxreqrt' times  *
                >131  *                                                       *
                >132  *********************************************************
                >133
BBE8: A9 50     >134  PKINCREQ lda   #r_PKINC   ; Send PEEKINC request
BBEA: D0 02     >135           bne   SIMPLREQ   ; (always)
                >136
BBEC: A9 58     >137  PKPOKREQ lda   #r_PKPOK   ; Send PEEKPOKE request
                >138
BBEE: 8D 3D B8  >139  SIMPLREQ sta   sbuf+rqmd  ; Save request type.
BBF1: A9 03     >140           lda   #maxreqrt  ; Set request retry
BBF3: 8D 54 B8  >141           sta   reqretry   ;  counter.
BBF6: AD 3D B8  >142  :retry   lda   sbuf+rqmd  ; Send request.
BBF9: 20 F3 BC  >143           jsr   REQUEST
BBFC: 90 06     >144           bcc   :done      ; Done if no error.
BBFE: CE 54 B8  >145           dec   reqretry   ; Dec request retry count
BC01: D0 F3     >146           bne   :retry     ; Try until OK or exhausted,
BC03: 38        >147           sec              ;  then return with C set.
BC04: 60        >148  :done    rts
```

```
                >152  ************************************************************
                >153  *                                                          *
                >154  *           P K I N C S R V ,   P K P O K S R V            *
                >155  *                                                          *
                >156  *             Michael J. Mahon - Nov 05, 2004              *
                >157  *                  Revised Apr 30, 2010                    *
                >158  *                                                          *
                >159  *                Copyright (c) 2004, 2008                  *
                >160  *                                                          *
                >161  *  Service machine 'rbuf+frm's request to send 2 bytes     *
                >162  *  at our 'rbuf+adr', then increment value by 'rbuf+len'   *
                >163  *  (PEEKINC) or store 'rbuf+len' as new value (PEEKPOKE).  *
                >164  *                                                          *
                >165  *  The PEEKINC and PEEKPOKE requests are "network atomic"  *
                >166  *  read-modify-write primitives for synchronization and    *
                >167  *  allocation operations.                                  *
                >168  *                                                          *
                >169  *  PKINCSRV and PKPOKSRV do the following steps:           *
                >170  *     1. Save initial 2-byte value in ACK buffer           *
                >171  *     2. If PKINCSRV: Increment the value by 'rbuf+len'    *
                >172  *     3. If PKPOKSRV: Set the value to 'rbuf+len'.         *
                >173  *     4. Send the ACK packet with the initial value.       *
                >174  *                                                          *
                >175  ************************************************************
                >176
BC05: A9 FF     >177  PKINCSRV lda   #$FF       ; Set mask to "increment"
BC07: D0 02     >178           bne   pkxxxsrv   ;  and go do it.
                >179
BC09: A9 00     >180  PKPOKSRV lda   #0         ; Set mask to "move"
BC0B: 85 EC     >181  pkxxxsrv sta   ckbyte
BC0D: 20 A2 BF  >182           jsr   ra=>a      ; Set up data address
BC10: A0 00     >183           ldy   #0
BC12: 18        >184           clc
BC13: B1 FC     >185  :movinc  lda   (address),y ; Get original value
BC15: 99 43 B8  >186           sta   sbuf+len,y ;   and save it for ACK.
BC18: 25 EC     >187           and   ckbyte     ; ($FF=inc, $00=move)
BC1A: 79 4B B8  >188           adc   rbuf+len,y
BC1D: 91 FC     >189           sta   (address),y ; Replace with new value.
BC1F: C8        >190           iny
BC20: 98        >191           tya              ; Don't disturb carry.
BC21: 49 02     >192           eor   #2         ; Done?
BC23: D0 EE     >193           bne   :movinc    ; -No, go again.
BC25: 4C 12 BE  >194           jmp   SENDACK    ; -Yes, send ACK with value.
```

```
>196  *-------------------------------------------------------*
>197  *              POKE, RUN, and BRUN Protocol             *
>198  *-------------------------------------------------------*
>199  *         Requester                    Server           *
>200  *  ======================     ======================  *
>201  *  xxxx   REQ (addr,leng)  ====>                       *
>202  *                          <==== POKE   ACK            *
>203  *                    Data  ====>                       *
>204  *                            :                         *
>205  *                    Data  ====>                       *
>206  *                          <==== POKE   DACK           *
>207  *-------------------------------------------------------*
```

```
                >211  ************************************************************
                >212  *                                                          *
                >213  *     P O K E R E Q ,  R U N R E Q ,   B R U N R E Q        *
                >214  *                                                          *
                >215  *            Michael J. Mahon - May 11, 1996               *
                >216  *                  Revised Sep 25, 2008                    *
                >217  *                                                          *
                >218  *             Copyright (c) 1996, 2004, 2008               *
                >219  *                                                          *
                >220  *  Request machine 'sbuf+dst' to store 'sbuf+len' bytes    *
                >221  *  at its 'sbuf+adr', and send them from our location      *
                >222  *  'locaddr'.                                              *
                >223  *                                                          *
                >224  *  These requests, like others, will retry the request     *
                >225  *  in case of error, up to 'maxreqrt' times.  If errors    *
                >226  *  persist, it will return with Carry set.                 *
                >227  *                                                          *
                >228  *  POKEREQ, RUNREQ, and BRUNREQ do the following steps:    *
                >229  *     1. Make the request (and receive the ACK)            *
                >230  *     2. Send 'sbuf+len' bytes of data from 'locaddr'      *
                >231  *     3. Receive DATA ACK response                         *
                >232  *     4. Retry in case of error up to 'maxreqrt' times     *
                >233  *                                                          *
                >234  ************************************************************
                >235
BC28: A9 60     >236  RUNREQ   lda   #r_RUN     ; Send RUN request.
BC2A: D0 06     >237           bne   setreq     ; (always)
                >238
BC2C: A9 68     >239  BRUNREQ  lda   #r_BRUN    ; Send BRUN request.
BC2E: D0 02     >240           bne   setreq     ; (always)
                >241
BC30: A9 10     >242  POKEREQ  lda   #r_POKE    ; Send POKE request.
BC32: 8D 3D B8  >243  setreq   sta   sbuf+rqmd  ; Set request code
BC35: A2 03     >244           ldx   #maxreqrt  ; Set request retry
BC37: 8E 54 B8  >245           stx   reqretry   ;  counter.
BC3A: AD 3D B8  >246  :retry   lda   sbuf+rqmd  ; Recover request code
BC3D: 20 F3 BC  >247           jsr   REQUEST
BC40: B0 0B     >248           bcs   :failed
BC42: 20 E9 BE  >249           jsr   lasl=>al   ; Set up address/length.
BC45: 20 B0 BF  >250           jsr   SENDLONG   ; Send multiple packets.
BC48: 20 4F BD  >251           jsr   RCVDACK    ; Receive DATA ACK packet.
BC4B: 90 06     >252           bcc   :done      ; -OK, return.
BC4D: CE 54 B8  >253  :failed  dec   reqretry   ; Dec request retry count
BC50: D0 E8     >254           bne   :retry     ; Try until OK or exhausted,
BC52: 38        >255           sec              ;  then return with C set.
BC53: 60        >256  :done    rts
```

```
                        >260  ************************************************************
                        >261  *                                                          *
                        >262  *    P O K E S R V ,   R U N S R V ,   B R U N S R V       *
                        >263  *                                                          *
                        >264  *             Michael J. Mahon - May 11, 1996              *
                        >265  *                   Revised Jan 24, 2008                   *
                        >266  *                                                          *
                        >267  *              Copyright (c) 1996, 2008, 2009              *
                        >268  *                                                          *
                        >269  *  Service machine 'rbuf+frm's request to poke 'rbuf+len'* *
                        >270  *  bytes of data to our 'rbuf+adr'.                        *
                        >271  *                                                          *
                        >272  *  If RUNSRV, initialize Applesoft and RUN the BASIC       *
                        >273  *  program transferred.  (Address must be > $800.)         *
                        >274  *                                                          *
                        >275  *  If BRUNSRV, CALL the code transferred with (A,X) set    *
                        >276  *  to the code's load address.                             *
                        >277  *                                                          *
                        >278  *  POKESRV, RUNSRV, and BRUNSRV do the following steps:    *
                        >279  *     1. If RUNSRV: lock net, save CSW/KSW hooks, and      *
                        >280  *        coldstart Applesoft.                              *
                        >281  *     2. Send the ACK packet                               *
                        >282  *     3. Receive multiple packets to 'rbuf+adr'            *
                        >283  *     4. If data received OK, send DATA ACK packet         *
                        >284  *     5. If BRUNSRV: Set (A,X) to address & JMP to code.   *
                        >285  *     6. If RUNSRV: Init Applesoft ptrs, fix up links,     *
                        >286  *        restore CSW/KSW hooks, and RUN the program.       *
                        >287  *                                                          *
                        >288  ************************************************************
                        >289
                        >291  savhooks equ   $2FC       ; Save hooks at end of page 2
BC54: C7 BC             >292  sethooks dw    rts        ; For COLDSTRT and FIXLINKS
BC56: 6D BC             >293           dw    POKESRV    ;  use hooks to retain control.
                        >294
BC58: 8D 5B C0          >295  RUNSRV   sta   dsend+1    ; Lock net for coldstart
BC5B: A2 03             >296           ldx   #3          ; Save and set CSW/KSW hooks
BC5D: B5 36             >297  :saveset lda   CSW,x       ;  to <rts,POKESRV> to retain
BC5F: 9D FC 02          >298           sta   savhooks,x ;   control after coldstart.
BC62: BD 54 BC          >299           lda   sethooks,x
BC65: 95 36             >300           sta   CSW,x
BC67: CA                >301           dex
BC68: 10 F3             >302           bpl   :saveset
BC6A: 4C 00 E0          >303           jmp   COLDSTRT   ;    BASIC coldstart.
                        >305  BRUNSRV  equ   *
BC6D: 20 12 BE          >306  POKESRV  jsr   SENDACK    ; ACK the request.
BC70: 20 98 BF          >307           jsr   rarl=>al   ; Set up address/length.
BC73: 20 CF BF          >308           jsr   RCVLONG    ; Receive long data message.
BC76: B0 4F             >309           bcs   rts        ; Receive error.
                        >310           delay 40         ; Allow requester to receive.
BC78: A2 08             >310           ldx   #40/5      ; (5 cycles per iteration)
BC7A: CA                >310  ]delay   dex
BC7B: D0 FD             >310           bne   ]delay
```

```
                >310            eom
BC7D: A9 03     >311            lda   #rm_DACK   ; Send DATA ACK
BC7F: 20 14 BE >312             jsr   SENDRSP    ;   packet.
BC82: AD 45 B8 >313             lda   rbuf+rqmd  ; Recover request
BC85: C9 11     >314            cmp   #r_POKE+rm_REQ ; POKE?
BC87: F0 3D     >315            beq   :ok        ; -Yes, return.
BC89: C9 69     >316            cmp   #r_BRUN+rm_REQ ; -No, BRUN?
BC8B: F0 5D     >317            beq   docall     ; -Yes, do CALL.
BC8D: AD CF 03 >321             lda   nadapage   ; -No, RUN.  Set HIMEM to
BC90: 85 74     >322            sta   HIMEM+1    ;   NadaNet load page.
BC92: 85 70     >323            sta   FRETOP+1
BC94: 18        >324            clc
BC95: AD 49 B8 >325             lda   rbuf+adr   ; Set PSTART to start
BC98: 85 67     >326            sta   PSTART     ;  addr and VARTAB to
BC9A: 6D 4B B8 >327             adc   rbuf+len   ;   end of program.
BC9D: 85 69     >328            sta   VARTAB
BC9F: AD 4A B8 >329             lda   rbuf+adr+1
BCA2: 85 68     >330            sta   PSTART+1
BCA4: 6D 4C B8 >331             adc   rbuf+len+1
BCA7: 85 6A     >332            sta   VARTAB+1
                >333            mov16 #:run;KSW  ; Retain control after
BCA9: A9 B4     >333            lda   #:run      ; Move 2 bytes
BCAB: 85 38     >333            sta   KSW
BCAD: A9 BC     >333            lda   #:run/$100 ; high byte of immediate
BCAF: 85 39     >333            sta   1+KSW
                >333            eom
BCB1: 4C F2 D4 >334             jmp   FIXLINKS   ;  fixing up prog links.
BCB4: A2 04     >335  :run      ldx   #4         ; Restore CSW/KSW hooks.
BCB6: BD FB 02 >336   :restore  lda   savhooks-1,x
BCB9: 95 35     >337            sta   CSW-1,x
BCBB: CA        >338            dex
BCBC: D0 F8     >339            bne   :restore
BCBE: 8A        >340            txa              ; Set byte preceding
BCBF: 81 B8     >341            sta   (TXTPTR,x) ;  program to zero,
BCC1: 85 D8     >342            sta   ONERR      ;   clear ONERR flag, and
BCC3: 4C 66 D5 >343             jmp   RUNPROG    ; RUN the Applesoft prog.
                >345
BCC6: 18        >346  :ok       clc              ; Good return.
BCC7: 60        >347  rts       rts              ; Return.
```

```
                >349  *-----------------------------------------------------*
                >350  *         Requester                    Server         *
                >351  *  ======================     ======================  *
                >352  *  BPOKE  REQ (addr,val)   ====>                      *
                >353  *                                (Broadcast, No ACK)   *
                >354  *-----------------------------------------------------*
                >355
                >358  *******************************************************
                >359  *                                                     *
                >360  *                    B P O K E R E Q                  *
                >361  *                                                     *
                >362  *            Michael J. Mahon - Nov. 04, 2004         *
                >363  *                  Revised Aug 20, 2008               *
                >364  *                                                     *
                >365  *                Copyright (c) 2004, 2008             *
                >366  *                                                     *
                >367  *  Broadcast request to all serving machines to store 2  *
                >368  *  bytes in 'sbuf+len' at address 'sbuf+adr'.         *
                >369  *                                                     *
                >370  *  BPOKE, unlike most requests, is broadcast, and so  *
                >371  *  is not acknowledged by any receiver.  To eliminate *
                >372  *  the chance of collision, it holds the bus locked for  *
                >373  *  20ms after arbitration, then sends the request packet.*
                >374  *  This allows enough time for any colliding sender to  *
                >375  *  send its request and re-arbitrate while the bus is  *
                >376  *  locked, so that there is no contention when the BPOKE *
                >377  *  request is finally sent.                           *
                >378  *                                                     *
                >379  *  BPOKEREQ does the following steps:                 *
                >380  *     1. Broadcast arbitrate and lock the bus         *
                >381  *     2. Set up BPOKE request                         *
                >382  *     3. Send the BPOKE request packet.               *
                >383  *                                                     *
                >384  *******************************************************
                >385
BCC8: 20 CB BD  >386  BPOKEREQ jsr   BCASTARB   ; Bcast arbitrate & lock bus
BCCB: A9 49     >387           lda   #r_BPOKE+rm_REQ ; Set up BPOKE request.
BCCD: 8D 3D B8  >388           sta   sbuf+rqmd
BCD0: 4C 26 BE  >389           jmp   SENDCTL    ; Send the request.
```

```
                >393  ************************************************************
                >394  *                                                          *
                >395  *                    B P O K E S R V                       *
                >396  *                                                          *
                >397  *            Michael J. Mahon - Nov 05, 2004               *
                >398  *                 Revised May 21, 2008                     *
                >399  *                                                          *
                >400  *               Copyright (c) 2004, 2008                   *
                >401  *                                                          *
                >402  *  Service machine 'rbuf+frm's request to poke 2 bytes     *
                >403  *  of data in 'rbuf+len' to our 'rbuf+adr'.                *
                >404  *                                                          *
                >405  *  BPOKESRV does the following:                            *
                >406  *     1. Move 'rbuf+len' to memory at 'rbuf+adr'.          *
                >407  *                                                          *
                >408  ************************************************************
                >409
BCD3: 20 A2 BF  >410  BPOKESRV jsr   ra=>a      ; Set up pointer
BCD6: A0 01     >411           ldy   #1         ; and move 2 bytes.
BCD8: B9 4B B8  >412  :move    lda   rbuf+len,y
BCDB: 91 FC     >413           sta   (address),y
BCDD: 88        >414           dey
BCDE: 10 F8     >415           bpl   :move
BCE0: 18        >416           clc
BCE1: 60        >417           rts             ; All done.
```

```
                >419  *-----------------------------------------------------*
                >420  *         Requester                    Server         *
                >421  *  ======================     ======================  *
                >422  *  CALL   REQ (addr,A,X)   ====>                      *
                >423  *                          <==== CALL   ACK           *
                >424  *-----------------------------------------------------*
                >425
                >428  *******************************************************
                >429  *                                                     *
                >430  *                    C A L L R E Q                    *
                >431  *                                                     *
                >432  *           Michael J. Mahon - May 11, 1996           *
                >433  *                 Revised Apr 30, 2010                *
                >434  *                                                     *
                >435  *           Copyright (c) 1996, 2004, 2010            *
                >436  *                                                     *
                >437  *  Request machine 'sbuf+dst' to call a subroutine at  *
                >438  *  address 'sbuf+adr' with parameters A = 'sbuf+len' and *
                >439  *  X = 'sbuf+len+1'.                                  *
                >440  *                                                     *
                >441  *  CALLREQ jumps to SIMPLREQ to retry the request in case*
                >442  *  of error, up to 'maxreqrt' times.  If errors persist, *
                >443  *  SIMPLREQ will return to the caller with Carry set.  *
                >444  *                                                     *
                >445  *  CALLREQ does the following steps:                  *
                >446  *     1. Make the CALL request (and receive the ACK)   *
                >447  *     2. Retry in case of error up to 'maxreqrt' times *
                >448  *                                                     *
                >449  *******************************************************
                >450
BCE2: A9 18     >451  CALLREQ  lda   #r_CALL    ; Send CALL request and
BCE4: 4C EE BB  >452           jmp   SIMPLREQ   ;  receive ACK (or error).
```

```
                >456  ************************************************************
                >457  *                                                          *
                >458  *                      C A L L S R V                       *
                >459  *                                                          *
                >460  *             Michael J. Mahon - May 11, 1996              *
                >461  *                   Revised Sep 25, 2008                   *
                >462  *                                                          *
                >463  *              Copyright (c) 1996, 2004, 2008              *
                >464  *                                                          *
                >465  *  Service machine 'rbuf+frm's request to call a           *
                >466  *  subroutine at our 'rbuf+adr' with parameters            *
                >467  *  A = 'rbuf+len' and X = 'rbuf+len+1'.  Flags are set      *
                >468  *  according to the value of A.                            *
                >469  *                                                          *
                >470  *  Note that when the subroutine returns, it returns to    *
                >471  *  whoever called SERVER.                                  *
                >472  *                                                          *
                >473  *  CALLSRV does the following steps:                       *
                >474  *     1. Send the ACK packet                               *
                >475  *     2. Load parameters from 'rbuf+len' into A and X      *
                >476  *     3. Call subroutine at 'rbuf+adr'                     *
                >477  *                                                          *
                >478  ************************************************************
                >479
BCE7: 20 12 BE  >480  CALLSRV  jsr   SENDACK    ; ACK the request.
BCEA: AE 4C B8  >481  docall   ldx   rbuf+len+1 ; Set X parameter
BCED: AD 4B B8  >482           lda   rbuf+len   ;  and A parameter, and
BCF0: 6C 49 B8  >483           jmp   (rbuf+adr) ; Jump to requested address.
```

```
                >487  ************************************************************
                >488  *                                                          *
                >489  *                      R E Q U E S T                       *
                >490  *                                                          *
                >491  *            Michael J. Mahon - April 20, 2004             *
                >492  *                  Revised Aug 17, 2008                    *
                >493  *                                                          *
                >494  *             Copyright (c) 1996, 2004, 2008               *
                >495  *                                                          *
                >496  *  Handle request protocol for the request in A & 'sbuf'.  *
                >497  *                                                          *
                >498  *  Retry the protocol for up to 'reqtime' ms. (up to       *
                >499  *  'retrylim' times).  If successful, return with valid    *
                >500  *  response in 'rbuf' and Carry clear.                      *
                >501  *                                                          *
                >502  *  If request timed out, return with Carry set and A=0.    *
                >503  *                                                          *
                >504  *  If NAK received, return with Carry set and A>0.         *
                >505  *                                                          *
                >506  *  REQUEST performs the following steps:                   *
                >507  *     1. Complete control pkt in 'sbuf' (request in A)     *
                >508  *     2. Arbitrate for the use of the bus                  *
                >509  *     3. Send the request specified in 'sbuf'              *
                >510  *     4. Receive the control response into 'rbuf'          *
                >511  *     5. Check 'rbuf' for a valid, expected response       *
                >512  *     6. Retry steps 2 to 5 up to 'retrylim' times         *
                >513  *     7. When ACKed, NAKed, or timed-out, return           *
                >514  *                                                          *
                >515  ************************************************************
                >516
BCF3: 09 01     >517  REQUEST  ora   #rm_REQ    ; Add REQ modifier and
BCF5: 8D 3D B8  >518           sta   sbuf+rqmd  ; Store request code.
BCF8: AD 4F B8  >519           lda   retrylim   ; Init retry counter.
BCFB: 8D 55 B8  >520           sta   retrycnt
BCFE: AD 55 B8  >521  :retry   lda   retrycnt   ; Timed out?
BD01: F0 4A     >522           beq   :err       ; -Yes, return w/ C set, A=0
BD03: CE 55 B8  >523           dec   retrycnt   ; Dec retry counter.
BD06: 20 00 BE  >524           jsr   ARBTRATE   ; Arbitrate for & lock bus
BD09: 20 26 BE  >525           jsr   SENDCTL    ; Send request in 'sbuf'.
BD0C: 20 00 BF  >526           jsr   RCVCTL     ; Receive response in 'rbuf'.
BD0F: 90 0D     >527           bcc   :ok        ; -Clean packet received.
                >528           dlyms reqdelay   ;  delay a few ms.
BD11: A0 11     >528           ldy   #reqdelay  ; Delay 1ms. per iteration
                >528  ]dly     delay 1020-4     ; Cycles per ms. - 4
BD13: A2 CB     >528           ldx   #1020-4/5  ; (5 cycles per iteration)
BD15: CA        >528  ]delay   dex
BD16: D0 FD     >528           bne   ]delay
                >528           eom
BD18: 88        >528           dey
BD19: D0 F8     >528           bne   ]dly
                >528           eom
BD1B: 4C FE BC  >529           jmp   :retry     ;    and try again...
```

```
                >530
BD1E: AD 47 B8  >531  :ok       lda   rbuf+dst   ; Message received, is
BD21: CD 3C B8  >532            cmp   self       ;  it for us?
BD24: D0 1D     >533            bne   :proterr   ; -No, error.
BD26: AD 3F B8  >534            lda   sbuf+dst   ; -Yes.  Is it from
BD29: CD 48 B8  >535            cmp   rbuf+frm   ;   our destination?
BD2C: D0 15     >536            bne   :proterr   ; -No.  Protocol error.
BD2E: AD 3D B8  >537            lda   sbuf+rqmd  ; -Yes.  Is the
BD31: 29 F8     >538            and   #reqmask   ;   modifier field
BD33: 09 02     >539            ora   #rm_ACK    ;    'ACK'?
BD35: CD 45 B8  >540            cmp   rbuf+rqmd  ;      as expected?
BD38: F0 0F     >541            beq   :good      ; -Yes, good response!
BD3A: 29 F8     >542            and   #reqmask   ; -No, construct
BD3C: 09 04     >543            ora   #rm_NAK    ;   the 'NAK' value.
BD3E: CD 45 B8  >544            cmp   rbuf+rqmd  ; Is it a NAK?
BD41: F0 08     >545            beq   :nakexit   ; -Yes, return w/ C set, A=1
BD43: 20 8F BF  >546  :proterr  jsr   PROTERR    ; -No, count protocol errors.
BD46: 4C FE BC  >547            jmp   :retry     ;    and try again...
                >548
BD49: 18        >549  :good     clc              ; Signal good ACK
BD4A: 60        >550            rts              ;  and return.
                >551
BD4B: A9 01     >552  :nakexit  lda   #1         ; Signal NAK
BD4D: 38        >553  :err      sec              ; Signal error
BD4E: 60        >554            rts              ;  and return.
```

```
                        >558  ************************************************************
                        >559  *                                                          *
                        >560  *                      R C V D A C K                       *
                        >561  *                                                          *
                        >562  *             Michael J. Mahon - Apr 19, 2004              *
                        >563  *                   Revised Aug 17, 2008                   *
                        >564  *                                                          *
                        >565  *                Copyright (c) 2004, 2008                  *
                        >566  *                                                          *
                        >567  *  Receive DATA ACK packet.  Require a good cksum,         *
                        >568  *  addressed to us, response req = sent req.  If all OK,   *
                        >569  *  return with Carry clear, else with Carry set.           *
                        >570  *                                                          *
                        >571  ************************************************************
                        >572
BD4F: 20 00 BF          >573  RCVDACK  jsr   RCVCTL     ; Receive response packet.
BD52: B0 1E             >574          bcs   :err        ; Cksum error or timeout.
BD54: AD 47 B8          >575  :ok      lda   rbuf+dst    ; Is packet for us?
BD57: CD 3C B8          >576          cmp   self
BD5A: D0 18             >577          bne   :proterr    ; -No, protocol error.
BD5C: AD 48 B8          >578          lda   rbuf+frm    ; Was it sent by receiver?
BD5F: CD 3F B8          >579          cmp   sbuf+dst
BD62: D0 10             >580          bne   :proterr    ; -No, protocol error.
BD64: AD 3D B8          >581          lda   sbuf+rqmd   ; Construct sent req
BD67: 29 F8             >582          and   #reqmask    ;  with the expected
BD69: 09 03             >583          ora   #rm_DACK    ;   'DACK' modifier.
BD6B: CD 45 B8          >584          cmp   rbuf+rqmd   ; Does it match?
BD6E: D0 04             >585          bne   :proterr    ; -No, protocol error.
BD70: 18                >586          clc               ; -Yes, clear Carry
BD71: 60                >587  :return  rts               ;   and return.
                        >588
BD72: D0 FD             >589  :err     bne   :return     ; Cksum error return.
BD74: 38                >590  :proterr sec               ; Return with C set,
BD75: 4C 8F BF          >591          jmp   PROTERR     ;  after counting error.
```

```
 60           put   PUTMGETM
>1    ************************************************************
>2    *                                                          *
>3    *                     Message Server                       *
>4    *                                                          *
>5    *            Michael J. Mahon - April 20, 2004             *
>6    *                  Revised May 21, 2008                    *
>7    *                                                          *
>8    *             Copyright (c) 2004, 2005, 2008               *
>9    *                                                          *
>10   *       Client Request Routines                            *
>11   *         Put Message Request                              *
>12   *         Get Message Request                              *
>13   *                                                          *
>14   *       Server Definitions                                 *
>15   *         Message Page Table                               *
>16   *         Message Class Table                              *
>17   *         Message Buffers (pages)                          *
>18   *                                                          *
>19   *       Server Routines (w/ Monitor)                       *
>20   *         Put Message Server                               *
>21   *         Get Message Server                               *
>22   *                                                          *
>23   *       Utility Routines                                   *
>24   *         Look Up class in Message Table                   *
>25   *                                                          *
>26   ************************************************************
>27
>28   *----------------------------------------------------------*
>29   *         Requester                    Server              *
>30   *  ========================     ======================     *
>31   *  PUTMSG REQ (class,leng) ====>                           *
>32   *         (lock)             :                            *
>33   *                          (<==== PUTMSG NAK if no space) *
>34   *                                                         *
>35   *                           <==== PUTMSG ACK              *
>36   *         Data < 256 bytes ====>                          *
>37   *                           <==== PUTMSG DACK             *
>38   *----------------------------------------------------------*
>39   *  GETMSG REQ (class)       ====>                         *
>40   *         (lock)             :                            *
>41   *                          (<==== GETMSG NAK if no msg)   *
>42   *                                                         *
>43   *                           <==== GETMSG ACK (class,leng) *
>44   *                           <==== Data < 256 bytes        *
>45   *  GETMSG DACK              ====>                         *
>46   *----------------------------------------------------------*
```

```
                >49   ************************************************************
                >50   *                                                          *
                >51   *                      P U T M R E Q                       *
                >52   *                                                          *
                >53   *            Michael J. Mahon - April 17, 2004             *
                >54   *                  Revised May 21, 2008                    *
                >55   *                                                          *
                >56   *                Copyright (c) 2004, 2008                  *
                >57   *                                                          *
                >58   *  Request message server (at 'sbuf+dest') to accept a     *
                >59   *  message of class 'sbuf+adr' and length 'sbuf+len'       *
                >60   *  at our local address 'locaddr'.                         *
                >61   *                                                          *
                >62   *  PUTMREQ will retry the request in case of timeout or    *
                >63   *  checksum errors up to 'maxreqrt' times.  If errors      *
                >64   *  persist, it returns with C set and A=0.                 *
                >65   *                                                          *
                >66   *  If the server NAKs the request for lack of space,       *
                >67   *  PUTMREQ returns with C set and A=1.                     *
                >68   *                                                          *
                >69   *  PUTMREQ does the following steps:                       *
                >70   *     1. Make the PUTMSG request                           *
                >71   *     2. If server NAKs, return with C set and A=1.        *
                >72   *     3. Send 'sbuf+len'-byte message from 'locaddr'       *
                >73   *     4. Receive DATA ACK packet                           *
                >74   *     5. Retry in case of error up to 'maxreqrt' times     *
                >75   *     6. If unsuccessful, return with C set and A=0.       *
                >76   *                                                          *
                >77   ************************************************************
                >78
BD78: A9 03     >79   PUTMREQ  lda   #maxreqrt  ; Set request retry
BD7A: 8D 54 B8  >80            sta   reqretry   ;  counter.
BD7D: A9 20     >81   :retry   lda   #r_PUTMSG  ; Send PUTMSG request.
BD7F: 20 F3 BC  >82            jsr   REQUEST
BD82: B0 0D     >83            bcs   :failed
BD84: 20 E9 BE  >84            jsr   lasl=>al   ; Set up address/length
BD87: 20 B0 BF  >85            jsr   SENDLONG   ;  and send message.
BD8A: 20 4F BD  >86            jsr   RCVDACK    ; Receive DATA ACK packet.
BD8D: 90 0A     >87            bcc   :done      ; -All OK.
BD8F: A9 00     >88            lda   #0         ; Not a NAK error
BD91: D0 05     >89   :failed  bne   :nakexit
BD93: CE 54 B8  >90   :cksumer dec   reqretry   ; Dec request retry count
BD96: D0 E5     >91            bne   :retry     ; Try until OK or exhausted,
BD98: 38        >92   :nakexit sec               ;  then return with C set.
BD99: 60        >93   :done    rts
```

```
                        >95   ************************************************************
                        >96   *                                                          *
                        >97   *                      G E T M R E Q                       *
                        >98   *                                                          *
                        >99   *            Michael J. Mahon - April 19, 2004             *
                        >100  *                 Revised May 21, 2008                     *
                        >101  *                                                          *
                        >102  *               Copyright (c) 2004, 2008                   *
                        >103  *                                                          *
                        >104  *  Request message server (at 'sbuf+dst') to deliver       *
                        >105  *  the first message of class 'sbuf+adr' to our address    *
                        >106  *  'locaddr', actual length in 'rbuf+len' after ACK.       *
                        >107  *                                                          *
                        >108  *  GETMREQ will retry the request in case of timeout or    *
                        >109  *  checksum errors up to 'maxreqrt' times.  If errors      *
                        >110  *  persist, it returns with C set and A=0.                 *
                        >111  *                                                          *
                        >112  *  If the server NAKs the request because the message      *
                        >113  *  queue is empty, GETMREQ returns with C set and A=1.     *
                        >114  *                                                          *
                        >115  *  GETMREQ does the following steps:                       *
                        >116  *     1. Make the GETMSG request                           *
                        >117  *     2. If server NAKs, return with C set and A=1.        *
                        >118  *     3. Receive 'rbuf+len'-byte message to 'locaddr'      *
                        >119  *     4. If no error, send DATA ACK packet                 *
                        >120  *     5. Retry in case of error up to 'maxreqrt' times     *
                        >121  *     6. If unsuccessful, return with C set and A=0.       *
                        >122  *                                                          *
                        >123  ************************************************************
                        >124
BD9A: A9 03             >125  GETMREQ  lda   #maxreqrt  ; Set request retry
BD9C: 8D 54 B8          >126           sta   reqretry   ;  counter.
BD9F: A9 28             >127  :retry   lda   #r_GETMSG  ; Send GETMSG request.
BDA1: 20 F3 BC          >128           jsr   REQUEST
BDA4: B0 1C             >129           bcs   :failed    ; Timeout or no msg.
BDA6: 20 F3 BE          >130           jsr   la=>a      ; Set up address
                        >131           mov16 rbuf+len;length ; and length.
BDA9: AD 4B B8          >131           lda   rbuf+len   ; Move 2 bytes
BDAC: 85 FE             >131           sta   length
BDAE: AD 4C B8          >131           lda   1+rbuf+len
BDB1: 85 FF             >131           sta   1+length
                        >131           eom
BDB3: 20 CF BF          >132           jsr   RCVLONG    ; Receive segmented message
BDB6: B0 0C             >133           bcs   :err       ; Timeout or cksum err.
                        >134           delay 40         ; Kill some time...
BDB8: A2 08             >134           ldx   #40/5      ; (5 cycles per iteration)
BDBA: CA                >134  ]delay   dex
BDBB: D0 FD             >134           bne   ]delay
                        >134           eom
BDBD: A9 03             >135           lda   #rm_DACK   ; -OK, send DATA ACK.
BDBF: 4C 14 BE          >136           jmp   SENDRSP    ;   and return w/ C clear.
                        >137
```

```
BDC2: D0 05    >138  :failed  bne   :nak       ; Server has no message.
BDC4: CE 54 B8 >139  :err     dec   reqretry   ; Cksum or timeout; dec count.
BDC7: D0 D6    >140           bne   :retry     ; Try until OK or exhausted,
BDC9: 38       >141  :nak     sec             ;  then return with C set.
BDCA: 60       >142           rts
```

```
 61           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   ************************************************************
```

```
                >48   ************************************************************
                >49   *                                                          *
                >50   *                     B C A S T A R B                      *
                >51   *                                                          *
                >52   *             Michael J. Mahon - Aug 20, 2008              *
                >53   *                                                          *
                >54   *                    Copyright (c) 2008                    *
                >55   *                                                          *
                >56   *  Broadcast Arbitrate is the precursor to any broadcast   *
                >57   *  request.  Since there are no ACKs from receivers, it    *
                >58   *  takes steps to ensure that it controls the network      *
                >59   *  and all receivers are ready to receive data:            *
                >60   *                                                          *
                >61   *  1. Arbitrate for and lock the network                   *
                >62   *  2. Delay 20ms. for any collisions to resolve and for    *
                >63   *     any slow pollers to reach their RCVPKT holds         *
                >64   *  3. Set 'sbuf+dst' to 0 for broadcast                    *
                >65   *                                                          *
                >66   ************************************************************
                >67
BDCB: 20 00 BE  >68   BCASTARB jsr   ARBTRATE   ; Arbitrate and lock network.
                >69            dlyms 20         ; Let collisions resolve.
BDCE: A0 14     >69            ldy   #20        ; Delay 1ms. per iteration
                >69   ]dly     delay 1020-4     ; Cycles per ms. - 4
BDD0: A2 CB     >69            ldx   #1020-4/5  ; (5 cycles per iteration)
BDD2: CA        >69   ]delay   dex
BDD3: D0 FD     >69            bne   ]delay
                >69            eom
BDD5: 88        >69            dey
BDD6: D0 F8     >69            bne   ]dly
                >69            eom
BDD8: A9 00     >70            lda   #0         ; Set broadcast
BDDA: 8D 3F B8  >71            sta   sbuf+dst   ;  request.
BDDD: 60        >72            rts              ;  and return.
```

```
                >76   ]end     align 256        ; Align to next page.
BDDE: 00 00 00  >76           ds    *-1/256*256+256-*
                >76           eom
                >77   xmain   equ   *-]end     ; (Timing-critical code)
                >78
                >79   ********************************************************
                >80   *                                                      *
                >81   *                    A R B T R A T E                   *
                >82   *                                                      *
                >83   *             Michael J. Mahon - May 1, 1996           *
                >84   *                 Revised Nov 05, 2004                 *
                >85   *                                                      *
                >86   *                   Copyright (c) 1996                 *
                >87   *                                                      *
                >88   *  Waits until bus has been idle for 'arbtime' plus     *
                >89   *  machine id # * 22 cycles, then locks bus and sends   *
                >90   *  the request control packet.                         *
                >91   *                                                      *
                >92   ********************************************************
                >93
BE00: AE 51 B8  >94   ARBTRATE ldx   arbxv      ; Set arbitration wait.
BE03: CD E8 C0  >95           cmp   zipslow    ; Zip Chip to 1MHz mode.
BE06: 2C 62 C0  >96   :waitidl bit   drecv      ; Wait for idle bus.
BE09: 30 F5     >97           bmi   ARBTRATE   ; Restart timing.
BE0B: CA        >98           dex
BE0C: D0 F8     >99           bne   :waitidl   ; ...not yet.
BE0E: 8D 5B C0 >100           sta   dsend+1    ; Got it!  Lock the bus
BE11: 60       >101           rts              ;  and return.
```

```
                >103  ************************************************************
                >104  *                                                          *
                >105  *                      S E N D P K T                       *
                >106  *                                                          *
                >107  *           Michael J. Mahon - April 15, 1996              *
                >108  *            Stephen Thomas - June 27, 2005                *
                >109  *                                                          *
                >110  *          Copyright (c) 1996, 2003, 2004, 2005            *
                >111  *                                                          *
                >112  *  Sends (X) bytes (1..256) starting at (A,Y) to the       *
                >113  *  currently selected machine(s).                          *
                >114  *                                                          *
                >115  *  SENDPKT does the following steps:                       *
                >116  *     1. Put Zip Chip in 'slow mode' for >38,000 cycles    *
                >117  *     2. Send start signal:  31 cyc ONE, 16 cyc ZERO,      *
                >118  *        8 cyc ONE, 8 cyc ZERO                             *
                >119  *     3. Send (X) data bytes (at 94-95 cyc/byte)           *
                >120  *     4. Send one check byte (95 cyc), leaves bus ZERO      *
                >121  *     5. Returns with Carry clear.                         *
                >122  *                                                          *
                >123  *  SENDCTL performs a SENDPKT on the control packet        *
                >124  *  send buffer 'sbuf'.                                     *
                >125  *                                                          *
                >126  *  SENDRSP builds a packet specified by A in 'sbuf'        *
                >127  *  for the request in 'rbuf', then sends it.               *
                >128  *                                                          *
                >129  *  SENDACK builds an ACK packet in 'sbuf' for the          *
                >130  *  request in 'rbuf', then sends it.                       *
                >131  *                                                          *
                >132  *  To obtain maximum sending speed (8 cycles/bit), the     *
                >133  *  inner loop of the actual sending code is unrolled       *
                >134  *  into a lattice, with two alternative straight-line      *
                >135  *  execution paths.  One of these sends an alternating     *
                >136  *  sequence of ones and zeroes; the other sends the        *
                >137  *  inverse alternating sequence.  Execution is bounced     *
                >138  *  from one path to the other depending on the data        *
                >139  *  being sent.  Branch-taken delays are compensated for    *
                >140  *  by the fact that branches are only necessary when no    *
                >141  *  change in bus state is required.                        *
                >142  *                                                          *
                >143  ************************************************************
                >144
BE12: A9 02     >145  SENDACK  lda   #rm_ACK    ; Build an ACK packet
BE14: 85 EC     >146  SENDRSP  sta   ckbyte     ; Store modifier for ora
BE16: AD 45 B8  >147           lda   rbuf+rqmd  ; Get received request
BE19: 29 F8     >148           and   #reqmask   ;  isolate request
BE1B: 05 EC     >149           ora   ckbyte     ;   and OR in modifier.
BE1D: 8D 3D B8  >150           sta   sbuf+rqmd  ; Set response code.
BE20: AD 48 B8  >151           lda   rbuf+frm
BE23: 8D 3F B8  >152           sta   sbuf+dst   ; Destination (= requester)
BE26: A9 3D     >153  SENDCTL  lda   #<sbuf     ; Control pkt send buffer
BE28: A0 B8     >154           ldy   #>sbuf
```

```
BE2A: A2 08    >155           ldx   #lenctl
               >156
BE2C: CD 70 C0 >160  SENDPKT  cmp   ptrig      ; Trigger paddle timer
BE2F: 8D 5B C0 >162           sta   dsend+1    ; Send start signal ONE
BE32: 20 9B BE >163           jsr   :exit      ; Stretch it.
BE35: 86 EC    >164           stx   ckbyte     ; Seed ckbyte with length.
BE37: 84 EE    >165           sty   ptr+1      ; Y = start address hi
BE39: A0 00    >166           ldy   #0         ; Index first data byte
BE3B: 18       >167           clc              ; Ensure C clear at exit
BE3C: 08       >168           php              ; Save interrupt state
BE3D: 78       >169           sei              ;  and disable interrupts.
               >170
               >171  * Time-critical region. Timings for :tnn labels and
               >172  * t= comments are relative to the preceding timing point
               >173  * (start sync or servo).
               >174
BE3E: 8D 5A C0 >175           sta   dsend+0    ; Send start sync ZERO
BE41: 2C 9B BE >176           bit   :exit      ; Set V to send ckbyte at end
BE44: 85 ED    >177           sta   ptr        ; A = start address lo
BE46: B1 ED    >178           lda   (ptr),y    ; Get first data (Y=0 so no px)
BE48: 8D 5B C0 >179           sta   dsend+1    ; Send coarse sync at t=16
BE4B: EA       >180           nop
BE4C: 38       >181           sec              ; Ensure C set between bytes
BE4D: 99 5A C0 >182           sta   dsend+0,y  ; Release coarse sync at t=24
BE50: B0 03    >183           bcs   :servo     ; Go send servo at t=32
               >184
BE52: C8       >185  :t84v0   iny              ; Get next data byte and
BE53: B1 ED    >186           lda   (ptr),y    ;  send servo at t=94 or 95
               >187
BE55: 8D 5B C0 >188  :servo   sta   dsend+1    ; Servo ONE
BE58: 2A       >189           rol
BE59: 90 41    >190           bcc   :t06b7v1
BE5B: 8D 5A C0 >191           sta   dsend+0    ; Bit 7 ZERO at t=8
BE5E: 2A       >192           rol
BE5F: B0 3D    >193           bcs   :t14b6v0
BE61: 8D 5B C0 >194           sta   dsend+1    ; Bit 6 ONE at t=16
BE64: 2A       >195  :t17b6v1 rol
BE65: 90 39    >196           bcc   :t22b5v1
BE67: 8D 5A C0 >197           sta   dsend+0    ; Bit 5 ZERO at t=24
BE6A: 2A       >198  :t25b5v0 rol
BE6B: B0 35    >199           bcs   :t30b4v0
BE6D: 8D 5B C0 >200           sta   dsend+1    ; Bit 4 ONE at t=32
BE70: 2A       >201  :t33b4v1 rol
BE71: 90 31    >202           bcc   :t38b3v1
BE73: 8D 5A C0 >203           sta   dsend+0    ; Bit 3 ZERO at t=40
BE76: 2A       >204  :t41b3v0 rol
BE77: B0 2D    >205           bcs   :t46b2v0
BE79: 8D 5B C0 >206           sta   dsend+1    ; Bit 2 ONE at t=48
BE7C: 2A       >207  :t49b2v1 rol
BE7D: 90 29    >208           bcc   :t54b1v1
BE7F: 8D 5A C0 >209           sta   dsend+0    ; Bit 1 ZERO at t=56
BE82: 2A       >210  :t57b1v0 rol
```

```
BE83: B0 25    >211           bcs   :t62b0v0
BE85: 8D 5B C0 >212           sta   dsend+1    ; Bit 0 ONE at t=64
BE88: 2A       >213  :t65b0v1 rol              ; Restore data, set C
BE89: EA       >214           nop
BE8A: 8D 5A C0 >215           sta   dsend+0    ; Idle/interbyte ZERO at t=72
               >216
BE8D: 45 EC    >217  :t73v0   eor   ckbyte     ; Compute checksum
BE8F: 85 EC    >218           sta   ckbyte     ;  and save it.
BE91: CA       >219           dex              ; Count bytes sent
BE92: D0 BE    >220           bne   :t84v0     ; Loop while more to send
               >221
BE94: 50 04    >222  :t83v0   bvc   :done      ; Quit if ckbyte already sent;
BE96: E8       >223           inx              ;  else count ckbyte,
BE97: B8       >224           clv              ;  clear send-ckbyte flag,
BE98: 50 BB    >225           bvc   :servo     ; Send ckbyte servo at t=95
               >226
BE9A: 28       >227  :done    plp              ; Restore int state
BE9B: 60       >228  :exit    rts              ;  and return with C clear.
               >229
BE9C: 90 1E    >230  :t06b7v1 bcc   :t09b7v1   ; These are all for timing
BE9E: B0 22    >231  :t14b6v0 bcs   :t17b6v0   ;  equalization and all of
BEA0: 90 26    >232  :t22b5v1 bcc   :t25b5v1   ;  them are always taken
BEA2: B0 2A    >233  :t30b4v0 bcs   :t33b4v0
BEA4: 90 2E    >234  :t38b3v1 bcc   :t41b3v1
BEA6: B0 32    >235  :t46b2v0 bcs   :t49b2v0
BEA8: 90 36    >236  :t54b1v1 bcc   :t57b1v1
BEAA: B0 3A    >237  :t62b0v0 bcs   :t65b0v0
               >238
BEAC: 90 B6    >239  :t14b6v1 bcc   :t17b6v1
BEAE: B0 BA    >240  :t22b5v0 bcs   :t25b5v0
BEB0: 90 BE    >241  :t30b4v1 bcc   :t33b4v1
BEB2: B0 C2    >242  :t38b3v0 bcs   :t41b3v0
BEB4: 90 C6    >243  :t46b2v1 bcc   :t49b2v1
BEB6: B0 CA    >244  :t54b1v0 bcs   :t57b1v0
BEB8: 90 CE    >245  :t62b0v1 bcc   :t65b0v1
BEBA: B0 D1    >246  :t70v0   bcs   :t73v0
               >247
BEBC: 2A       >248  :t09b7v1 rol
BEBD: 90 ED    >249           bcc   :t14b6v1
BEBF: 8D 5A C0 >250           sta   dsend+0    ; Bit 6 ZERO at t=16
BEC2: 2A       >251  :t17b6v0 rol
BEC3: B0 E9    >252           bcs   :t22b5v0
BEC5: 8D 5B C0 >253           sta   dsend+1    ; Bit 5 ONE at t=24
BEC8: 2A       >254  :t25b5v1 rol
BEC9: 90 E5    >255           bcc   :t30b4v1
BECB: 8D 5A C0 >256           sta   dsend+0    ; Bit 4 ZERO at t=32
BECE: 2A       >257  :t33b4v0 rol
BECF: B0 E1    >258           bcs   :t38b3v0
BED1: 8D 5B C0 >259           sta   dsend+1    ; Bit 3 ONE at t=40
BED4: 2A       >260  :t41b3v1 rol
BED5: 90 DD    >261           bcc   :t46b2v1
BED7: 8D 5A C0 >262           sta   dsend+0    ; Bit 2 ZERO at t=48
```

```
BEDA: 2A       >263  :t49b2v0 rol
BEDB: B0 D9    >264           bcs   :t54b1v0
BEDD: 8D 5B C0 >265           sta   dsend+1    ; Bit 1 ONE at t=56
BEE0: 2A       >266  :t57b1v1 rol
BEE1: 90 D5    >267           bcc   :t62b0v1
BEE3: 8D 5A C0 >268           sta   dsend+0    ; Bit 0 ZERO at t=64
BEE6: 2A       >269  :t65b0v0 rol              ; Restore data, set C
BEE7: B0 D1    >270           bcs   :t70v0     ; Always taken
```

```
                >272  ************************************************************
                >273  *                                                          *
                >274  *                    L A S L = > A L                       *
                >275  *                                                          *
                >276  *                       L A = > A                          *
                >277  *                                                          *
                >278  ************************************************************
                >279
                >280  lasl=>al mov16 sbuf+len;length ; 'sbuf' length -> length
BEE9: AD 43 B8  >280           lda   sbuf+len   ; Move 2 bytes
BEEC: 85 FE     >280           sta   length
BEEE: AD 44 B8  >280           lda   1+sbuf+len
BEF1: 85 FF     >280           sta   1+length
                >280           eom
                >281  la=>a    mov16 locaddr;address ; Local address -> address
BEF3: AD 4D B8  >281           lda   locaddr    ; Move 2 bytes
BEF6: 85 FC     >281           sta   address
BEF8: AD 4E B8  >281           lda   1+locaddr
BEFB: 85 FD     >281           sta   1+address
                >281           eom
BEFD: 60        >282           rts
```

```
                >284  ]end     align 256        ; Align to next page.
BEFE: 00 00     >284           ds    *-1/256*256+256-*
                >284           eom
                >285  xsend    equ   *-]end     ; (Timing-critical code)
                >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
BF00: A9 45     >405  RCVCTL   lda   #<rbuf     ; Receive control pkt to 'rbuf'
BF02: A0 B8     >406           ldy   #>rbuf
BF04: A2 08     >407           ldx   #lenctl
BF06: 85 ED     >408  RCVPKT   sta   ptr        ; A = buf address lo
BF08: 84 EE     >409           sty   ptr+1      ; Y = buf address hi
BF0A: 8A        >410  RCVPTR   txa              ; Seed checksum with length
BF0B: CA        >411           dex              ; X = length 1..256 (0=>256);
BF0C: 86 EB     >412           stx   lastidx    ;  convert to last buffer index
                >413
BF0E: AC 52 B8  >414           ldy   tolim      ; Wait <= (tolim-1) * 2.8ms.
BF11: A2 5C     >415           ldx   #arbx      ;  plus minimum arb time.
BF13: 08        >416           php              ; Save interrupt state
BF14: 78        >417           sei              ;  and disable interrupts.
BF15: 2C E8 C0  >418           bit   zipslow    ; Slow any Zip Chip to 1 MHz.
                >419
BF18: 2C 62 C0  >420  :waitstr bit   drecv      ; Wait for starting ONE.
BF1B: 30 0A     >421           bmi   :gotstr
BF1D: CA        >422           dex              ; (inner loop is 11 cycles)
BF1E: D0 F8     >423           bne   :waitstr   ; Keep waiting...
BF20: 88        >424           dey              ; (outer loop is 2820 cycles)
BF21: D0 F5     >425           bne   :waitstr   ; Loop for 'timeout' ms.
                >426
BF23: 28        >427           plp              ; Restore int state
BF24: 98        >428           tya              ; Signal timeout (A=0, Z set)
BF25: 38        >429           sec              ;  and return with C set.
BF26: 60        >430  :exit    rts
                >431
BF27: 2C E8 C0  >432  :gotstr  bit   zipslow    ; Slow Zip Chip for packet.
                >433
BF2A: 2C 62 C0  >434  :waitsyn bit   drecv      ; Wait for 16-cycle sync ZERO;
BF2D: 30 FB     >435           bmi   :waitsyn   ;  too bad if bus locks forever!
                >436
BF2F: A0 FF     >437           ldy   #$FF       ; Index-1 of first data location
BF31: A2 7F     >438           ldx   #$7F       ; CPX #0-7F sets C, 80-FF clears
```

```
BF33: EC 62 C0 >439  :synrt07 cpx   drecv      ; Check for coarse sync at rt=10
BF36: B0 05    >440           bcs   :synrt14   ; Only do delay if still ZERO
               >441
BF38: 2C 26 BF >442  :synrt19 bit   :exit      ; Set V (not-ckbyte flag)
BF3B: 70 04    >443           bvs   :servo     ; Do first servo check at rt=29
               >444
BF3D: 18       >445  :synrt14 clc             ; 6-cycle coarse sync delay
BF3E: 90 F8    >446           bcc   :synrt19   ; (1 extra to get here, 5 back)
               >447
BF40: B8       >448  :rt88    clv             ; Clear not-ckbyte flag
               >449
BF41: EC 62 C0 >450  :servo   cpx   drecv      ; Check for servo transition
BF44: 90 02    >451  :rt01    bcc   :rt04      ; Delay 3 cyc if past servo,
BF46: EA       >452           nop             ;  6 if not
BF47: EA       >453           nop
               >454
BF48: 85 EC    >455  :rt04    sta   ckbyte     ; Update checksum
BF4A: C8       >456           iny             ; Index next data location
               >457
BF4B: EC 62 C0 >458  :rt09    cpx   drecv      ; C <-- ~ bit 7 at rt=12
BF4E: 2A       >459           rol             ; Shift bit 7 in.
BF4F: EA       >460           nop
BF50: EC 62 C0 >461           cpx   drecv      ; C <-- ~ bit 6 at rt=20
BF53: 2A       >462           rol
BF54: EA       >463           nop
BF55: EC 62 C0 >464           cpx   drecv      ; C <-- ~ bit 5 at rt=28
BF58: 2A       >465           rol
BF59: EA       >466           nop
BF5A: EC 62 C0 >467           cpx   drecv      ; C <-- ~ bit 4 at rt=36
BF5D: 2A       >468           rol
BF5E: EA       >469           nop
BF5F: EC 62 C0 >470           cpx   drecv      ; C <-- ~ bit 3 at rt=44
BF62: 2A       >471           rol
BF63: EA       >472           nop
BF64: EC 62 C0 >473           cpx   drecv      ; C <-- ~ bit 2 at rt=52
BF67: 2A       >474           rol
BF68: EA       >475           nop
BF69: EC 62 C0 >476           cpx   drecv      ; C <-- ~ bit 1 at rt=60
BF6C: 2A       >477           rol
BF6D: EA       >478           nop
BF6E: EC 62 C0 >479           cpx   drecv      ; C <-- ~ bit 0 at rt=68
BF71: 2A       >480  :rt69    rol
BF72: 50 0A    >481  :rt71    bvc   :rcvdone   ; quit after ckbyte
               >482
BF74: 91 ED    >483  :rt73    sta   (ptr),y    ; Save data (always 6cy)
BF76: 45 EC    >484  :rt79    eor   ckbyte     ; Compute checksum
BF78: C4 EB    >485  :rt82    cpy   lastidx    ; Stored last byte?
BF7A: F0 C4    >486  :rt85    beq   :rt88      ; Go clear not-ckbyte flag if so
BF7C: D0 C3    >487  :rt87    bne   :servo     ; Do next servo sample at rt=93
               >488
BF7E: 45 EC    >489  :rcvdone eor   ckbyte     ; A = 0 if ckbyte = sum
BF80: F0 08    >490           beq   :goodck    ; -No error.
```

```
                >491            inc16 ckerr       ; Count checksum error.
BF82: EE 58 B8  >491            inc   ckerr       ; Increment 16-bit word.
BF85: D0 03     >491            bne   *+5         ; - No carry.
BF87: EE 59 B8  >491            inc   ckerr+1     ; Propagate carry.
                >491            eom
BF8A: 28        >492  :goodck   plp               ; Restore int state
BF8B: C9 01     >493            cmp   #1          ; Set C & NZ if checksum bad,
BF8D: AA        >494            tax               ;  clear C and set Z if good
BF8E: 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.
BF8F: EE 56 B8  >503           inc   errprot    ; Increment 16-bit word.
BF92: D0 03     >503           bne   *+5        ; - No carry.
BF94: EE 57 B8  >503           inc   errprot+1  ; Propagate carry.
                >503           eom
BF97: 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
BF98: AD 4B B8  >515           lda   rbuf+len   ; Move 2 bytes
BF9B: 85 FE     >515           sta   length
BF9D: AD 4C B8  >515           lda   1+rbuf+len
BFA0: 85 FF     >515           sta   1+length
                >515           eom
                >516  ra=>a    mov16 rbuf+adr;address; 'rbuf' address -> address
BFA2: AD 49 B8  >516           lda   rbuf+adr   ; Move 2 bytes
BFA5: 85 FC     >516           sta   address
BFA7: AD 4A B8  >516           lda   1+rbuf+adr
BFAA: 85 FD     >516           sta   1+address
                >516           eom
BFAC: 60        >517           rts
```

```
                >520  ************************************************************
                >521  *                                                          *
                >522  *                    S E N D L O N G                       *
                >523  *                                                          *
                >524  *             Michael J. Mahon - May 5, 1996               *
                >525  *                  Revised May 21, 2008                    *
                >526  *                                                          *
                >527  *                Copyright (c) 1996, 2008                  *
                >528  *                                                          *
                >529  *  SENDLONG sends 'length' bytes from 'address' to the     *
                >530  *  currently selected machine(s).                          *
                >531  *                                                          *
                >532  *  DSENDLNG delays X*5-1 cycles and falls into SENDLONG.   *
                >533  *                                                          *
                >534  *  It segments a "message" longer than 256 bytes into a    *
                >535  *  series of 256-byte packets, plus a final packet         *
                >536  *  with the remainder of the data.  Each message packet    *
                >537  *  is sent with 'SENDPKT'.                                 *
                >538  *                                                          *
                >539  *  SENDLONG does not detect any errors.                    *
                >540  *                                                          *
                >541  ************************************************************
                >542
BFAD: CA        >543  DSENDLNG dex              ; Delay 5 * X - 1 cycles
BFAE: D0 FD     >544           bne   DSENDLNG   ;  and fall into SENDLONG.
BFB0: A5 FF     >545  SENDLONG lda   length+1   ; How many 256-byte pages?
BFB2: F0 0F     >546           beq   :short     ; - None, just a short pkt.
BFB4: A2 00     >547  :loop    ldx   #0         ; Set 256 byte packet.
BFB6: A5 FC     >548           lda   address    ;  and point to
BFB8: A4 FD     >549           ldy   address+1  ;   data buffer.
BFBA: 20 2C BE  >550           jsr   SENDPKT    ; Send 256 bytes.
BFBD: E6 FD     >551           inc   address+1  ; Advance to next page
BFBF: C6 FF     >552           dec   length+1   ;  and decrement page
BFC1: D0 F1     >553           bne   :loop      ;   count until done...
BFC3: A6 FE     >554  :short   ldx   length     ; Remaining data length.
BFC5: F0 07     >555           beq   :done      ; -All done.
BFC7: A5 FC     >556           lda   address
BFC9: A4 FD     >557           ldy   address+1
BFCB: 20 2C BE  >558           jsr   SENDPKT    ; Send the final packet.
BFCE: 60        >559  :done    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
BFCF: A5 FF     >585  RCVLONG  lda   length+1   ; How many 256-byte pages?
BFD1: F0 11     >586           beq   :short     ; - None, just a short pkt.
BFD3: A2 00     >587  :loop    ldx   #0          ; Set 256 byte packet.
BFD5: A5 FC     >588           lda   address     ;  and point to
BFD7: A4 FD     >589           ldy   address+1   ;   data buffer.
BFD9: 20 06 BF  >590           jsr   RCVPKT      ; Receive 256 bytes.
BFDC: B0 14     >591           bcs   :err        ; Receive error detected.
BFDE: E6 FD     >592           inc   address+1   ; Advance to next page
BFE0: C6 FF     >593           dec   length+1    ;  and decrement page
BFE2: D0 EF     >594           bne   :loop       ;   count until done...
BFE4: A6 FE     >595  :short   ldx   length      ; Remaining data length.
BFE6: F0 09     >596           beq   :done       ; -All done.
BFE8: A5 FC     >597           lda   address
BFEA: A4 FD     >598           ldy   address+1
BFEC: 20 06 BF  >599           jsr   RCVPKT      ; Receive final packet.
BFEF: B0 01     >600           bcs   :err        ; Keep track of any errors.
BFF1: 60        >601  :done    rts
                >602
BFF2: D0 03     >603  :err     bne   :ckerr      ; Checksum error.
BFF4: 20 8F BF  >604           jsr   PROTERR     ; Count protocol error.
BFF7: A8        >605  :ckerr   tay               ; Set Z flag from A.
BFF8: 60        >606           rts
```

```
                62   ]end     align 256        ; Align to page boundary
BFF9: 00 00 00  62            ds    *-1/256*256+256-*
                62            eom
                63   xreceive equ   *-]end     ; Extra space at end.
                64            err   *-1-entry/SIZE ; Can't exceed limit


--End assembly, 2304 bytes, Errors: 0


Symbol table - alphabetical order:

   @        =$B83B      ADDON   =$D998      AMPNADA =$B934      AMPVECT =$03F5
   ARBTRATE=$BE00       AX      =$48        BCASTARB=$BDCB      BCASTREQ=$BB81
   BELL     =$FF3A      BOOT2   =$B703      BOOTREQ =$BB7D      BPOKEREQ=$BCC8
   BPOKESRV=$BCD3       BRUNREQ =$BC2C      BRUNSRV =$BC6D      CALLREQ =$BCE2
   CALLSRV =$BCE7       CALL_t  =$8C        CHRGET  =$B1        CHRGOT  =$B7
   COLDSTRT=$E000       COUT    =$FDED      CROUT1  =$FD8B      CSW     =$36
   DSENDLNG=$BFAD       ERROR   =$D412      FAC     =$9D        FIXLINKS=$D4F2
   FLO2     =$EBA0      FORPNT  =$85        FRETOP  =$6F        FRMNUM  =$DD67
   GETADR   =$E752      GETBYT  =$E6F8      GETMREQ =$BD9A      GET_t   =$BE
   HIMEM    =$73     ?  HOME    =$FC58      INIT    =$BA97      INSTALL =$B909
   INTFLG   =$12        KSW     =$38        ONERR   =$D8        PEEKREQ =$BB94
   PEEKSRV  =$BBC5      PEEK_t  =$E2        PKINCREQ=$BBE8      PKINCSRV=$BC05
   PKPOKREQ=$BBEC       PKPOKSRV=$BC09      POKEREQ =$BC30      POKESRV =$BC6D
   POKE_t   =$B9     ?  PRBL2   =$F94A      PRBYTE  =$FDDA      PREAD   =$FB1E
?  PROGEND  =$AF        PROTERR =$BF8F      PSTART  =$67        PTRGET  =$DFE3
   PUTMREQ  =$BD78      PWREDUP =$03F4   ?  RARL=>AL=$B836      RCVCTL  =$BF00
   RCVDACK  =$BD4F      RCVLONG =$BFCF      RCVPKT  =$BF06      RCVPTR  =$BF0A
   REQUEST  =$BCF3      ROMboot =$00        RUNPROG =$D566      RUNREQ  =$BC28
   RUNSRV   =$BC58      RUN_t   =$AC        SENDACK =$BE12      SENDCTL =$BE26
   SENDLONG=$BFB0       SENDPKT =$BE2C      SENDRSP =$BE14      SERVER  =$BAE6
   SETFOR   =$EB27      SIMPLREQ=$BBEE      SIZE    =$0800      SOFTEV  =$03F2
   SYNCHR   =$DEC0      SYNERR  =$DEC9      TXTPTR  =$B8        VALTYP  =$11
   VARTAB   =$69     ?  VBL     =$C019   V? ]PROTERR=$BB51   V  ]cpx    =$0B
V  ]cpy     =$0B04   V  ]cy     =$4FB0   MV ]delay  =$BDD2   MV ]dly    =$BDD0
V? ]doit    =$BB83   V  ]end    =$BFF9   V  ]servpad=$FF        addr    =$46
   address  =$FC        adr     =$04     MD align   =$8000      an0     =$C058
   an1      =$C05A      an2     =$C05C   ?  an3     =$C05E      arbtime =$01
   arbx     =$5C        arbxv   =$B851   ?  bcast   =$B81E      bootself=$03CC
?  bpoke    =$B821   ?  brun    =$B82D      byte    =$00     ?  call    =$B815
   chain    =$B94A      ckbyte  =$EC        ckerr   =$B858      class   =$46
   cmd      =$B941      cmdptr  =$EC        cmdsave =$ED        cmdtable=$B864
   comp     =$B94D      crate   =$01        cyperms =$03FC   MD delay   =$8000
   dest     =$04        disp    =$EF     MD dlyms   =$8000      docall  =$BCEA
?  dos      =$00        drecv   =$C062      dsend   =$C05A      dsk6off =$C0E8
   dst      =$02        enhboot =$00        entry   =$B800   ?  ep      =$B700
   errprot  =$B856      errstop =$B85F      frm     =$03        frmc    =$01
   frmcerr  =$B85A   ?  gapwait =$07     ?  getmsg  =$B81B      idletime=$14
   idleto   =$08        idtable =$B85D      idtbl   =$BA5E   MD inc16   =$8000
   incr     =$48     ?  init    =$B809      instald =$B85D      iter    =$15
   kbstrobe=$C010       keybd   =$C000      la=>a   =$BEF3      lasl=>al=$BEE9
```

```
   lastidx =$EB        len     =$06        lenctl  =$08        length  =$FE
   lngth   =$48        lngth?  =$D0        locaddr =$B84D      locadr  =$52
   master  =$00     ?  maxarb  =$03        maxgap  =$57        maxreq  =$70
   maxreqrt=$03        maxretry=$32        modmask =$07     MD mov16   =$8000
   mserve  =$00        n60ms   =$14        nadapage=$03CF   ?  nadaver =$B85C
   nparms  =$B85E      null    =$BA5C      parmsiz =$15        pb0     =$C061
   pb1     =$C062      pb2     =$C063   ?  peek    =$B80F   ?  peekinc =$B824
?  peekpoke=$B827      pkxxxsrv=$BC0B   ?  poke    =$B812      ptr     =$ED
   ptrig   =$C070   ?  putmsg  =$B818      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   =$60        ra=>a   =$BFA2
   rarl=>al=$BF98      rbuf    =$B845   ?  rcvctl  =$B830   ?  rcvlong =$B839
?  rcvptr  =$B833      reqctr  =$B853      reqdelay=$11        reqdur  =$06
   reqfac  =$08        reqmask =$F8        reqpidle=$03        reqretry=$B854
   reqtime =$0BB8      reqto   =$01        retrycnt=$B855      retrylim=$B84F
   rm_ACK  =$02        rm_DACK =$03        rm_NAK  =$04        rm_REQ  =$01
   rqmd    =$00        rqperiod=$14        rts     =$BCC7   ?  run     =$B82A
   savhooks=$02FC      sbuf    =$B83D      self    =$B83C   ?  serve   =$B80C
   servecnt=$B850      servegap=$3A        servelp =$B803      service =$BA6A
   sethooks=$BC54      setid   =$BAC9      setreq  =$BC32   ?  spkr    =$C030
   svrxkbd =$BAE3   ?  t_BASIC =$E0     ?  t_SYNTH =$F0     ?  t_VOICE =$F1
   timeout =$BA52      tolim   =$B852      val     =$48        val?    =$D0
   var     =$80        varadr  =$B862      varcmd  =$B860      vartype =$B861
   verlen  =$13        vermsg  =$BA84      version =$31        warmstrt=$03CD
   word    =$40     ?  xboot   =$88     ?  xmain   =$22     ?  xreceive=$07
?  xsend   =$02        zipslow =$C0E8
```

Symbol table - numerical order:

```
   master  =$00     ?  dos     =$00        mserve  =$00        ROMboot =$00
   enhboot =$00        rqmd    =$00        byte    =$00        crate   =$01
   arbtime =$01        reqto   =$01        frmc    =$01        rm_REQ  =$01
   dst     =$02        rm_ACK  =$02     ?  xsend   =$02     ?  maxarb  =$03
   reqpidle=$03        maxreqrt=$03        frm     =$03        rm_DACK =$03
   adr     =$04        rm_NAK  =$04        dest    =$04        reqdur  =$06
   len     =$06     ?  gapwait =$07        modmask =$07     ?  xreceive=$07
   idleto  =$08        lenctl  =$08        reqfac  =$08        r_PEEK  =$08
V  ]cpx    =$0B        r_POKE  =$10        reqdelay=$11        VALTYP  =$11
   INTFLG  =$12        verlen  =$13        idletime=$14        rqperiod=$14
   n60ms   =$14        parmsiz =$15        iter    =$15        r_CALL  =$18
   r_PUTMSG=$20     ?  xmain   =$22        r_GETMSG=$28        r_GETID =$30
   version =$31        maxretry=$32        CSW     =$36        KSW     =$38
   r_BOOT  =$38        servegap=$3A        r_BCAST =$40        word    =$40
   addr    =$46        class   =$46        r_BPOKE =$48        lngth   =$48
   AX      =$48        incr    =$48        val     =$48        r_PKINC =$50
   locadr  =$52        maxgap  =$57        r_PKPOK =$58        arbx    =$5C
   r_RUN   =$60        PSTART  =$67        r_BRUN  =$68        VARTAB  =$69
   FRETOP  =$6F        maxreq  =$70        HIMEM   =$73        var     =$80
   FORPNT  =$85     ?  xboot   =$88        CALL_t  =$8C        FAC     =$9D
   RUN_t   =$AC     ?  PROGEND =$AF        CHRGET  =$B1        CHRGOT  =$B7
```

```
   TXTPTR  =$B8       POKE_t  =$B9       GET_t   =$BE       lngth?  =$D0
   val?    =$D0       ONERR   =$D8     ?  t_BASIC =$E0       PEEK_t  =$E2
   lastidx =$EB       ckbyte  =$EC       cmdptr  =$EC       ptr     =$ED
   cmdsave =$ED       disp    =$EF     ?  t_SYNTH =$F0     ?  t_VOICE =$F1
   reqmask =$F8       address =$FC       length  =$FE     V  ]servpad=$FF
   savhooks=$02FC     bootself=$03CC     warmstrt=$03CD     nadapage=$03CF
   SOFTEV  =$03F2     PWREDUP =$03F4     AMPVECT =$03F5     cyperms =$03FC
   SIZE    =$0800   V ]cpy     =$0B04     reqtime =$0BB8   V ]cy      =$4FB0
MD align   =$8000   MD dlyms   =$8000   MD delay   =$8000   MD mov16   =$8000
MD inc16   =$8000   ?  ep      =$B700     BOOT2   =$B703     entry   =$B800
   servelp =$B803   ?  init    =$B809   ?  serve   =$B80C   ?  peek    =$B80F
?  poke    =$B812   ?  call    =$B815   ?  putmsg  =$B818   ?  getmsg  =$B81B
?  bcast   =$B81E   ?  bpoke   =$B821   ?  peekinc =$B824   ?  peekpoke=$B827
?  run     =$B82A   ?  brun    =$B82D   ?  rcvctl  =$B830   ?  rcvptr  =$B833
?  RARL=>AL=$B836   ?  rcvlong =$B839     @       =$B83B     self    =$B83C
   sbuf    =$B83D     rbuf    =$B845     locaddr =$B84D     retrylim=$B84F
   servecnt=$B850     arbxv   =$B851     tolim   =$B852     reqctr  =$B853
   reqretry=$B854     retrycnt=$B855     errprot =$B856     ckerr   =$B858
   frmcerr =$B85A   ?  nadaver =$B85C     idtable =$B85D     instald =$B85D
   nparms  =$B85E     errstop =$B85F     varcmd  =$B860     vartype =$B861
   varadr  =$B862     cmdtable=$B864     INSTALL =$B909     AMPNADA =$B934
   cmd     =$B941     chain   =$B94A     comp    =$B94D     timeout =$BA52
   null    =$BA5C     idtbl   =$BA5E     service =$BA6A     vermsg  =$BA84
   INIT    =$BA97     setid   =$BAC9     svrxkbd =$BAE3     SERVER  =$BAE6
V? ]PROTERR=$BB51     BOOTREQ =$BB7D     BCASTREQ=$BB81   V? ]doit   =$BB83
   PEEKREQ =$BB94     PEEKSRV =$BBC5     PKINCREQ=$BBE8     PKPOKREQ=$BBEC
   SIMPLREQ=$BBEE     PKINCSRV=$BC05     PKPOKSRV=$BC09     pkxxxsrv=$BC0B
   RUNREQ  =$BC28     BRUNREQ =$BC2C     POKEREQ =$BC30     setreq  =$BC32
   sethooks=$BC54     RUNSRV  =$BC58     BRUNSRV =$BC6D     POKESRV =$BC6D
   rts     =$BCC7     BPOKEREQ=$BCC8     BPOKESRV=$BCD3     CALLREQ =$BCE2
   CALLSRV =$BCE7     docall  =$BCEA     REQUEST =$BCF3     RCVDACK =$BD4F
   PUTMREQ =$BD78     GETMREQ =$BD9A     BCASTARB=$BDCB   MV ]dly    =$BDD0
MV ]delay  =$BDD2     ARBTRATE=$BE00     SENDACK =$BE12     SENDRSP =$BE14
   SENDCTL =$BE26     SENDPKT =$BE2C     lasl=>al=$BEE9     la=>a   =$BEF3
   RCVCTL  =$BF00     RCVPKT  =$BF06     RCVPTR  =$BF0A     PROTERR =$BF8F
   rarl=>al=$BF98     ra=>a   =$BFA2     DSENDLNG=$BFAD     SENDLONG=$BFB0
   RCVLONG =$BFCF   V ]end    =$BFF9     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     ERROR   =$D412     FIXLINKS=$D4F2
   RUNPROG =$D566     ADDON   =$D998     FRMNUM  =$DD67     SYNCHR  =$DEC0
   SYNERR  =$DEC9     PTRGET  =$DFE3     COLDSTRT=$E000     GETBYT  =$E6F8
   GETADR  =$E752     SETFOR  =$EB27     FLO2    =$EBA0   ?  PRBL2   =$F94A
   PREAD   =$FB1E   ?  HOME    =$FC58     CROUT1  =$FD8B     PRBYTE  =$FDDA
   COUT    =$FDED     BELL    =$FF3A
```