```
*************************************************************
*                                                           *
*                    N A D A . P R O D O S                  *
*                                                           *
*                 ProDOS version of NadaNet                 *
*                                                           *
*             Michael J. Mahon - April 14, 1996             *
*                   Revised Jan 24, 2009                    *
*                                                           *
*    Copyright (c) 1996, 2003, 2004, 2005, 2008, 2009       *
*                                                           *
*   NadaNet is a suite of 6502 machine code routines        *
*   to support bidirectional communication among several    *
*   Apple // computers.  It uses one wire (plus ground)     *
*   connected to the game ports of the machines.            *
*                                                           *
*   An annunciator output is used to "broadcast" to all     *
*   machines, and a "pushbutton" input is used to sense     *
*   the state of the shared signalling wire.  This is       *
*   similar to Ethernet, but at lower speed and at TTL      *
*   levels.                                                 *
*                                                           *
*   The raw signaling speed is 1 bit every 8 cycles, or     *
*   127.6 kilobaud.  With byte separator overhead of 31     *
*   cycles, this translates to 1 byte every 94-95 cycles,   *
*   or over 10K bytes/sec (thanks to Stephen Thomas!).      *
*                                                           *
*   All signal transmission and reception is done with      *
*   precisely timed software routines.  Synchronization     *
*   is assured by a digital PLL at the receiver which       *
*   adapts to variations in timing of 93-96 cycles/byte.    *
*   (If a machine has a Zip Chip accelerator installed,     *
*   it is temporarily slowed during packet transmission     *
*   and reception.)                                         *
*                                                           *
*************************************************************

******************** Version setup ***********************

SIZE     equ   $900       ; "Do not exceed" size

         org   $9A00-SIZE ; 'NADANET' for ProDOS hosts
master   equ   1          ; Include master-only functions
dos      equ   0          ; Non-DOS version
crate    equ   0          ; Non-Crate version
mserve   equ   0          ; Non-Message Server version
ROMboot  equ   0          ; Non-ROM version
enhboot  equ   0          ; Non-Enhanced //e ROM version
```

```
 50           put   NADAHIST
>1    ************************************************************
>2    *                                                          *
>3    *                     Change History                       *
>4    *                                                          *
>5    *   01/24/09:                                              *
>6    *                                                          *
>7    *   Added ONERR ($D8) definition and code to clear the     *
>8    *   flag at boot time and at &RUN time.                    *
>9    *                                                          *
>10   *   01/06/09:                                              *
>11   *                                                          *
>12   *   Modified NADA.CRATE's NADABOOT2 to coldstart BASIC      *
>13   *   and set HIMEM to base of NadaNet.                      *
>14   *                                                          *
>15   *   Changed 'version' to a 2-digit BCD value that is       *
>16   *   used in messages and the version byte.                 *
>17   *                                                          *
>18   *   11/11/08:                                              *
>19   *                                                          *
>20   *   Modified RUNSRV to save and restore CSW/KSW hooks      *
>21   *   so that &RUN works properly with or without an OS.     *
>22   *                                                          *
>23   *   11/03/08:                                              *
>24   *                                                          *
>25   *   Added call to FIXLINKS into RUNSRV code to allow       *
>26   *   BASIC programs to be &RUN at any address > $800.       *
>27   *                                                          *
>28   *   10/06/08:                                              *
>29   *                                                          *
>30   *   Added simple BCAST action to SERVER that just sets     *
>31   *   'address' and 'length' to request values and then      *
>32   *   returns to the calling code to deal with the data.     *
>33   *                                                          *
>34   *   Added table of BCAST tags to NADADEFS to serve as a    *
>35   *   central directory of BCAST tag values.                 *
>36   *                                                          *
>37   *   09/25/08:                                              *
>38   *                                                          *
>39   *   Added &RUN and &BRUN, as derivatives of &POKE, to      *
>40   *   run Applesoft programs and M/L programs.               *
>41   *                                                          *
>42   *   Added RCVCTL, RCVPTR, rarl=>al, and RCVLONG to the     *
>43   *   entry point vector for use by BCAST server code.       *
>44   *                                                          *
>45   *   09/04/08:                                              *
>46   *                                                          *
>47   *   Restructured SERVER to correct failure to re-sync if   *
>48   *   'reqctr' was satisfied, and to minimze "deaf" time     *
>49   *   when iterating in SERVER.                              *
>50   *                                                          *
>51   *   Added 'reqpidle' (requests per idletime) definition,   *
>52   *   which is closer to typical.                            *
```

```
>53   *                                                               *
>54   *   08/20/08:                                                   *
>55   *                                                               *
>56   *   Added BCAST request as a general mechanism for              *
>57   *   broadcasting data.                                          *
>58   *                                                               *
>59   *   Added BCASTARB to arbitrate and lock net, then delay        *
>60   *   for 20ms. to allow all collisions to resolve and any        *
>61   *   "slow" pollers to get into their RCVCTL holds.  This        *
>62   *   arbitration will precede all broadcast requests, like       *
>63   *   BOOT, BCAST, and BPOKE.                                     *
>64   *                                                               *
>65   *   08/16/08:                                                   *
>66   *                                                               *
>67   *   Changed control packet format:                              *
>68   *   - Combined 'req' and 'mod' bytes into 'rqmd' byte.          *
>69   *   - Added complement of 'frm', called 'frmc', as a way        *
>70   *     to detect collisions of synchronized packets.             *
>71   *   - Removed "delayed BOOTREQ after GETID" boot protocol       *
>72   *   - Modified BOOTREQ to send old format packet for            *
>73   *     compatibility with v2.1 PassiveBoot ROM.                  *
>74   *   - Changed sign-on version to v3.0.                          *
>75   *                                                               *
>76   *   Moved error counters to just after IDTBL, and               *
>77   *   prefixed them with NadaNet version in hex.                  *
>78   *                                                               *
>79   *   05/21/08:                                                   *
>80   *                                                               *
>81   *   Made numerous significant space optimizations:              *
>82   *   - Added subroutines to set 'address' & 'length'             *
>83   *     from most common variables.                               *
>84   *   - Added PROTERR subroutine to increment count.              *
>85   *   - Put checksum counting inside RCVPKT.                      *
>86   *   - Added variable delay preceding SENDLONG.                  *
>87   *   - Placed all of the above in "slack" space following        *
>88   *     page-aligned SENDPKT and RCVPKT.                          *
>89   *   - Deleted MONITOR (can always load when needed).            *
>90   *                                                               *
>91   *   Fixed potential bug if INSTALL called >255 times.           *
>92   *                                                               *
>93   *   Added ID error checking to 'setid' and INIT.                *
>94   *                                                               *
>95   *   Changed broadcast BOOTREQ to not be protocol error.         *
>96   *                                                               *
>97   *   Added 'xmain', 'xsend', and 'xreceive' symbols to           *
>98   *   make slack space easy to read in symbol table.              *
>99   *                                                               *
>100  *   04/21/08:                                                   *
>101  *                                                               *
>102  *   Added 'svrxkbd' entry to SERVER, used by 'servelp'          *
>103  *   to ignore keyboard input.                                   *
>104  *                                                               *
>105  *   04/15/08:                                                   *
```

```
>106  *                                                               *
>107  *   Split NADADEFS include file into three parts so that        *
>108  *   the control packet definitions could be used without        *
>109  *   generating code for the vectors and variables.              *
>110  *                                                               *
>111  *   Added new "Enhanced boot" protocol for AppleCrate           *
>112  *   machines using Enhanced //e's.  The new protocol            *
>113  *   blindly broadcasts the boot code whenever &BOOTCODE         *
>114  *   is invoked.                                                 *
>115  *                                                               *
>116  *   Since only RCVPKT and RCVLONG, plus RESET and the           *
>117  *   actual boot logic need reside in ROM, it fits easily        *
>118  *   into the 2 pages of code used by the Enhanced //e's         *
>119  *   self-test code.                                             *
>120  *                                                               *
>121  *   To support this, a new conditional assembly flag,           *
>122  *   'enhboot' has been added and used to select the             *
>123  *   code in SENDRCV and NADADEFS for the new boot ROM.           *
>124  *                                                               *
>125  *   The new AppleCrate boot image will be prefixed with         *
>126  *   a second-stage boot that will use GETID to allocate         *
>127  *   unique machine IDs.  The DACK length hi-byte sent by        *
>128  *   Enhanced machines contains a "magic number" ($A5) to        *
>129  *   signal that GETID need not schedule a boot code send.       *
>130  *                                                               *
>131  *   The second stage boot will set up the page 3 RESET          *
>132  *   vector to go directly to NadaNet INIT, so it does           *
>133  *   not take up space in the running machine.                   *
>134  *                                                               *
>135  *   The second-stage boot code uses a non-zero 'bootself'       *
>136  *   value to indicate that a GETID has already been done        *
>137  *   and the second-stage can be skipped for unenhanced          *
>138  *   AppleCrate machines.                                        *
>139  *                                                               *
>140  *   The maximum number of machines has been increased to        *
>141  *   31, and the zeroth entry in the IDTABLE is 31.              *
>142  *                                                               *
>143  *   06/29/05:                                                   *
>144  *                                                               *
>145  *   Added NadaNet version 2.0 sign-on message to INIT.          *
>146  *                                                               *
>147  *   Replaced tree-like data send routing in SENDPKT with        *
>148  *   new, shorter, lattice-like routine created by Stephen       *
>149  *   Thomas.  The new routine sends all bits in uniform          *
>150  *   cells of 8 cycles, lowering the cycles/byte to 95           *
>151  *   from 106, for a speed increase of more than 11%.            *
>152  *                                                               *
>153  *   The new data transfer rate is over 10 KB/second.            *
>154  *                                                               *
>155  *   The packet start synchronization now allows RCVPKT          *
>156  *   to establish fine sync for the first byte.                  *
>157  *                                                               *
>158  *   The new RCVPKT samples data bitcells only during the        *
```

```
>159  *   5th, 6th, and 7th of 8 cycles, making NadaNet much   *
>160  *   more tolerant of long network time constants caused  *
>161  *   by too much cable or too high a pulldown resistance. *
>162  *                                                        *
>163  *   The timing of the check byte is now identical to the *
>164  *   timing of data bytes.                                *
>165  *                                                        *
>166  *   RCVPKT is also changed to reflect the new timings,   *
>167  *   which required unrolling the receive code again.     *
>168  *                                                        *
>169  *   Increased receive-to-send turnaround delays in       *
>170  *   PEEKSRV and GETMSRV to allow some margin for the     *
>171  *   receiving machine to start polling.                  *
>172  *                                                        *
>173  *   06/08/05:                                            *
>174  *                                                        *
>175  *   Fixed bus fight in RCVPKT pointed out by an astute   *
>176  *   reader of the code, Stephen Thomas, who also sent    *
>177  *   replacement code which only reads the paddle input   *
>178  *   and which cleverly combines data shifting with loop  *
>179  *   control, permitting the receive code to be re-rolled *
>180  *   to save space!                                       *
>181  *                                                        *
>182  *   12/01/04:                                            *
>183  *                                                        *
>184  *   Fixed GETID bug that left 'sbuf+adr' unset if the ID *
>185  *   received was not a temporary ID.  (For masters only) *
>186  *                                                        *
>187  *   Parameterized maximum number of machines (maxid) and *
>188  *   changed GETID so that any ID > maxid is considered a *
>189  *   temporary ID to be assigned a permanent ID.          *
>190  *                                                        *
>191  *   Changed handling of protocol errors in SERVER so     *
>192  *   they are "timed" as if they were requests.           *
>193  *                                                        *
>194  *   11/17/04:                                            *
>195  *                                                        *
>196  *   Changed 'servegap' wait time to 3/4 of min arb time  *
>197  *   to allow some margin for server routine processing    *
>198  *   after network is released (which is subtracted from  *
>199  *   "SERVER visible" inter-request gap).                 *
>200  *                                                        *
>201  *   11/13/04                                             *
>202  *                                                        *
>203  *   Changed AmperNada handler to leave return variable   *
>204  *   unchanged when an error occurs.                      *
>205  *                                                        *
>206  *   11/12/04:                                            *
>207  *                                                        *
>208  *   Increased BPOKE locked wait time to 20ms. to allow   *
>209  *   more "dead time" in an Applesoft &SERVE polling loop.*
>210  *                                                        *
>211  *   11/10/04:                                            *
```

```
>212  *                                                               *
>213  *   Changed SERVER gap wait to wait for an unchanging            *
>214  *   net, not an idle net, so that a BPOKE is received            *
>215  *   when preceded by a locked state.                             *
>216  *                                                               *
>217  *   Fixed bug in PKINCSRV that dropped carry.                    *
>218  *                                                               *
>219  *   11/08/04:                                                   *
>220  *                                                               *
>221  *   Changed AmperNada handler to throw an Applesoft             *
>222  *   "DATA" (49) error by default when a command fails.          *
>223  *   This error can be caught by an active ONERR, or it          *
>224  *   can be suppressed by appending a "#" to the command.        *
>225  *   If the error is suppressed, it is the programmer's          *
>226  *   responsibility to check status by PEEKing 1 and 0.          *
>227  *                                                               *
>228  *   11/06/04:                                                   *
>229  *                                                               *
>230  *   Removed &ONERR(err?) because its residual effects--         *
>231  *   storing status into variable memory--outlast any            *
>232  *   running program unless explicitly cancelled.  Using         *
>233  *   PEEK(1) is a safe and effective alternative solution.       *
>234  *                                                               *
>235  *   11/05/04:                                                   *
>236  *                                                               *
>237  *   Added BPOKE & PEEKINC requestors and servers.               *
>238  *                                                               *
>239  *   Added &IDTBL(val?) to retrieve address of 'idtable'         *
>240  *   in 'master' version.                                        *
>241  *                                                               *
>242  *   Changed ARBTRATE to use a single loop, and SETID to         *
>243  *   use ID for 'arbxv' when a temp ID (>$7F) is used.           *
>244  *                                                               *
>245  *   11/01/04:                                                   *
>246  *                                                               *
>247  *   Integrated AmperNada ampersand interface for BASIC          *
>248  *   into NadaNet.  Size limit is now $900.                      *
>249  *                                                               *
>250  *   10/27/04:                                                   *
>251  *                                                               *
>252  *   Fixed latent BOOT timing bug in server.                     *
>253  *                                                               *
>254  *   Changed SERVER so that it returns after processing          *
>255  *   any request, in addition to when a key is pressed.          *
>256  *                                                               *
>257  *   Changed SERVER and CALLSRV to do indirect jumps,            *
>258  *   rather than pushing addresses on stack for rts.             *
>259  *                                                               *
>260  *   Changed REQUEST resend count so that the request            *
>261  *   timeout set by 'reqtime' is accurate.                       *
>262  *                                                               *
>263  *   Changed MONITOR to wait for a minimum period of             *
>264  *   unchanging network state, rather than '0' state, so         *
```

```
>265  *   a locked state between packets is detected as a gap. *
>266  *                                                        *
>267  *   10/20/04:                                            *
>268  *                                                        *
>269  *   Added changes so that NADABOOT could be built using  *
>270  *   standard NADANET "put" files.                        *
>271  *                                                        *
>272  *   Split this change history into a separate file.      *
>273  *                                                        *
>274  *   10/18/04:                                            *
>275  *                                                        *
>276  *   Added code to INIT to set up $3CD with warm start    *
>277  *   'JMP servelp', so $3CF is NADANET's load page.       *
>278  *                                                        *
>279  *   10/13/04:                                            *
>280  *                                                        *
>281  *   Made PUTMREQ and GETMREQ conditional upon 'master'   *
>282  *   conditional compile flag.  PUTMSRV and GETMSRV are   *
>283  *   conditional upon 'not master'.  This frees up space  *
>284  *   for additional enhancements by splitting NadaNet into*
>285  *   different functional subsets for different purposes. *
>286  *                                                        *
>287  *   Made RCVPKT timeout variable so it can be set to the *
>288  *   minimum arbitration time while within a protocol,    *
>289  *   to protect the protocol from "outside" interference, *
>290  *   and set to 20ms. outside a protocol, when SERVE or   *
>291  *   MONITOR is running, to reduce polling dead time.     *
>292  *                                                        *
>293  *   Moved packet-starting 'ONE' earlier in SENDPKT so    *
>294  *   that ARBTRATE and RCVPKT do not need to double-poll  *
>295  *   bus to detect start pulse.                           *
>296  *                                                        *
>297  *   Changed BOOTREQ to use boot code address, length,    *
>298  *   and local address set up prior to SERVER call.       *
>299  *                                                        *
>300  *   Split entry point vector and variable definitions    *
>301  *   out into NADADEFS "put" file for use in other progs. *
>302  *                                                        *
>303  *   10/05/04:                                            *
>304  *                                                        *
>305  *   Changed ARBTRATE to lock the bus after a successful  *
>306  *   poll, so that the arbitration "increment" could be   *
>307  *   reduced to 22 cycles from 66.                        *
>308  *                                                        *
>309  *   Changed SETID arbitration time calculation to match. *
>310  *                                                        *
>311  *   Changed SERVER so that received requests, whether    *
>312  *   acted upon or not, are counted as 1/8 of a 20ms.     *
>313  *   timeout interval.  This allows time-related events   *
>314  *   to occur properly whether the net is busy or idle.   *
>315  *                                                        *
>316  *   Moved 'sbuf', 'rbuf', and counters so that they will *
>317  *   move less in the future.                             *
```

```
>318  *                                                               *
>319  *   Made REQUEST retry limit an initialized variable so         *
>320  *   that it can be lowered to speed up detection of a           *
>321  *   possibly non-existent machine.                              *
>322  *                                                               *
>323  *   Moved the 'monch' table used by PUTMSRV and GETMSRV         *
>324  *   from internal memory to the unused top of the page          *
>325  *   map table, saving 48 bytes of program memory.               *
>326  *                                                               *
>327  *   06/23/04:                                                   *
>328  *                                                               *
>329  *   Added iteration counter to SERVER and dispensed with        *
>330  *   SERVE1.  Changed SERVE sync wait to min arbitration         *
>331  *   time.                                                       *
>332  *                                                               *
>333  *   Disabled interrupts during SENDPKT and RCVPKT.              *
>334  *                                                               *
>335  *   Made "packet"/"message" nomenclature consistent.            *
>336  *                                                               *
>337  *   06/04/04:                                                   *
>338  *                                                               *
>339  *   Made INIT, SERVE, and BOOT functions and PEEK, POKE,        *
>340  *   and CALL functions separate include modules.                *
>341  *                                                               *
>342  *   Added text graphics for protocols.                          *
>343  *                                                               *
>344  *   05/26/04                                                    *
>345  *                                                               *
>346  *   Added MONITOR function for snooping all packets on          *
>347  *   the network and logging the first 8 bytes in memory.        *
>348  *                                                               *
>349  *   05/15/04:                                                   *
>350  *                                                               *
>351  *   Added "master" conditional assembly switch to               *
>352  *   control newly added  boot functions, BOOTREQ and            *
>353  *   GETIDSRV.                                                   *
>354  *                                                               *
>355  *   05/06/04:                                                   *
>356  *                                                               *
>357  *   Shortened arbitration time to 1 ms., since almost           *
>358  *   all protocols have less than 1 ms. delay between            *
>359  *   packets.  Exceptions will "lock" the net by pulling         *
>360  *   it high until they can respond.  This is effectively        *
>361  *   an extended "start" pulse, and RCVPKT will wait             *
>362  *   indefinitely for the transition to low.                     *
>363  *                                                               *
>364  *   Currently, only PUTMSRV and GETMSRV can take longer         *
>365  *   than 1 ms. to respond with ACK or NAK, so they must         *
>366  *   lock the net until their response.                          *
>367  *                                                               *
>368  *   The delay from last arbitration poll until beginning        *
>369  *   of "start" pulse is 54 cycles, so to avoid collision        *
>370  *   the arbitration delay difference between machines           *
```

```
>371  *   must exceed 54.  Since it must be a multiple of 11   *
>372  *   cycles, the offset is machine ID * 66 cycles.        *
>373  *                                                        *
>374  *   Since 8-byte data packets are indistinguishable      *
>375  *   from control packets, and since data packets are     *
>376  *   never delayed from a preceding control packet by     *
>377  *   more than 0.5 ms., SERVER must wait for a net "idle" *
>378  *   (low) state for 0.5 ms. to ensure that the next pkt  *
>379  *   it receives is not data.  This delay is only needed  *
>380  *   if the net has not been polled for more than the     *
>381  *   arbitration delay (~1 ms.).                          *
>382  *                                                        *
>383  *   04/19/04:                                            *
>384  *                                                        *
>385  *   Changed RCVPKT timeout to 20 ms., since responses    *
>386  *   are expected in much less.  SERVER loop results in   *
>387  *   "blind" time of less than 0.04 ms. per iteration.    *
>388  *                                                        *
>389  *   Changed CALLSRV to pass parameters A and X passed    *
>390  *   in 'rbuf+len' bytes.                                 *
>391  *                                                        *
>392  *   Factored RCVDACK code out for general use.           *
>393  *                                                        *
>394  *   Changed REQUEST to return on ACK or NAK response.    *
>395  *                                                        *
>396  *   03/05/04:                                            *
>397  *                                                        *
>398  *   Changed code to avoid address modification (to allow *
>399  *   code to run in ROM).  This adds one cycle/byte.      *
>400  *                                                        *
>401  *   Changed RCVPKT digital PLL back to +/- 2 cycles/byte *
>402  *   because of page crossing variances in SENDPKT and    *
>403  *   RCVPKT LDA and STA ops that had been overlooked.     *
>404  *                                                        *
>405  *   09/04/03:                                            *
>406  *                                                        *
>407  *   Changed SENDPKT and RCVPKT to new bit timing by      *
>408  *   unrolling loops.                                     *
>409  *                                                        *
>410  *   Changed RCVPKT digital PLL to be +/- 1 cycle/byte    *
>411  *   instead of +/- 2 cycles/byte to tighten tolerances.  *
>412  *                                                        *
>413  **********************************************************
```

```
 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    *                        v3.0                              *
                >5    *                                                          *
                >6    *            Michael J. Mahon - Oct 13, 2004               *
                >7    *                 Revised Oct 06, 2008                     *
                >8    *                                                          *
                >9    *               Copyright (c) 2004, 2008                   *
                >10   *                                                          *
                >11   ************************************************************
                >12
                >13   version  equ   $30        ; NadaNet version 3.0
                >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
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_RUN    ds    reqfac     ;   RUN request
0060: 00 00 00  >46   r_BRUN   ds    reqfac     ;   BRUN request
                >47                             ; ========================
                >48   maxreq   ds    0          ;   Max request + reqfac
                >49            dend
                >50
                >51            dum   1          ; Modifier codes (0 invalid):
0001: 00        >52   rm_REQ   ds    1          ;   Request
```

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

```
                 54           put   NADAVECTOR
                >1    ******************** Entry Points ************************
                >2
9100: 20 1C 92  >4    entry    jsr   INSTALL    ; BOOT entry: init and
9103: 20 F4 93  >5    servelp  jsr   svrxkbd    ;  SERVE ignoring keypresses
9106: 4C 03 91  >6             jmp   servelp    ;   forever...
                >7
9109: 4C 1C 92  >8    init     jmp   INSTALL    ; Initialize and return
910C: 4C F7 93  >9    serve    jmp   SERVER     ; Run request server
910F: 4C E1 94  >11   peek     jmp   PEEKREQ
9112: 4C 69 95  >12   poke     jmp   POKEREQ
9115: 4C 1B 96  >13   call     jmp   CALLREQ
9118: 4C BF 96  >14   putmsg   jmp   PUTMREQ
911B: 4C E1 96  >15   getmsg   jmp   GETMREQ
911E: 4C 97 94  >16   bcast    jmp   BCASTREQ
9121: 4C 01 96  >17   bpoke    jmp   BPOKEREQ
9124: 4C 35 95  >18   peekinc  jmp   PKINCREQ
9127: 4C 61 95  >19   run      jmp   RUNREQ
912A: 4C 65 95  >20   brun     jmp   BRUNREQ
912D: 4C 00 99  >21   rcvctl   jmp   RCVCTL
9130: 4C 0A 99  >22   rcvptr   jmp   RCVPTR
9133: 4C 98 99  >23   RARL=>AL jmp   rarl=>al
9136: 4C CF 99  >24   rcvlong  jmp   RCVLONG
                >41
                 55           put   NADAVARS
                >1    ********** Parameters and variables ************
                >2
                >6
9139: 00        >7    self     db    0          ; Our own machine ID
913A: 00 00 00  >8    sbuf     ds    lenctl     ; Control pkt send buffer
9142: 00 00 00  >9    rbuf     ds    lenctl     ; Control pkt receive buffer
914A: 00 00     >10   locaddr  dw    0          ; Local address of req data
914C: 32        >11   retrylim db    maxretry   ; Limit of REQUEST resends
914D: 00        >12   servecnt db    0          ; SERVE iterations (0=256)
                >13
                >14   parmsiz  equ   *-self     ; Size of parameter area
                >15
                >16   ************ Counters and Version **************
                >17
914E: 5C        >18   arbxv    db    arbx       ; Arbitrate X iters (modified)
914F: 01        >19   tolim    db    reqto      ; RCVPKT timeout limit
9150: 03        >20   reqctr   db    reqpidle   ; SERVER request counter
9151: 00        >21   reqretry db    0          ; xxxREQ retries remaining
9152: 00        >22   retrycnt db    0          ; REQUEST resend count
9153: 00 00     >23   errprot  dw    0          ; Protocol error count
9155: 00 00     >24   ckerr    dw    0          ; Checksum error count
9157: 00 00     >25   frmcerr  dw    0          ; 'frmc' collision errors
9159: 30        >26   nadaver  db    version    ; NadaNet version
                >27
                >29   * Table of allocated machine IDs (allocated = non-zero)
                >30
                >31   maxid    equ   31         ; Maximum number of machines
                >32
```

```
915A: 1F 04    >33   idtable  db    maxid,4+dos ; Table of machine attributes
915C: 00 00 00 >34            ds    maxid-1    ; Rest of ID table (=0)
               >39
                56           put   AMPERSAND
```

```
>2    ************************************************************
>3    *                                                          *
>4    *                    A M P E R N A D A                     *
>5    *                                                          *
>6    *            Michael J. Mahon - Oct 25, 2004               *
>7    *                 Revised Sep 25, 2008                     *
>8    *                                                          *
>9    *               Copyright (c) 2004, 2008                   *
>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
917A: 00        >56   instald  db    0          ; Installed flag
917B: 00        >57   nparms   db    0          ; # of parms seen
917C: 00        >58   errstop  db    0          ; "Throw error" flag
917D: 00        >59   varcmd   db    0          ; var parm descriptor
917E: 00        >60   vartype  db    0          ; variable type
917F: 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
                >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
9181: 53 45 52  >95   cmdtable asc   'SERVE',00                 ; &SERVE
9187: 15 00     >96            db    iter,0
9189: F7 93     >97            da    SERVER
                >98
918B: 50 55 54  >99            asc   'PUTMSG',00               ; &PUTMSG
9192: 04 46 48 >100            db    dest,class,lngth,locadr,0
9197: BF 96    >101            da    PUTMREQ
               >102
9199: BE 4D 53 >103            db    GET_t,'M','S','G',0       ; &GETMSG
919E: 04 46 D0 >104            db    dest,class,lngth?,locadr,0
91A3: E1 96    >105            da    GETMREQ
               >106
91A5: E2 49 4E >107            db    PEEK_t,'I','N','C',0      ; &PEEKINC
91AA: 04 46 48 >108            db    dest,addr,incr,val?,0
91AF: 35 95    >109            da    PKINCREQ
               >110
91B1: E2 00    >111            db    PEEK_t,0                  ; &PEEK
91B3: 04 46 48 >112            db    dest,addr,lngth,locadr,0
91B8: E1 94    >113            da    PEEKREQ
               >114
91BA: B9 00    >115            db    POKE_t,0                  ; &POKE
```

```
91BC: 04 46 48 >116           db    dest,addr,lngth,locadr,0
91C1: 69 95    >117           da    POKEREQ
               >118
91C3: AC 00    >119           db    RUN_t,0                    ; &RUN
91C5: 04 46 48 >120           db    dest,addr,lngth,locadr,0
91CA: 61 95    >121           da    RUNREQ
               >122
91CC: 42 AC 00 >123           db    'B',RUN_t,0                ; &BRUN
91CF: 04 46 48 >124           db    dest,addr,lngth,locadr,0
91D4: 65 95    >125           da    BRUNREQ
               >126
91D6: 8C 00    >127           db    CALL_t,0                   ; &CALL
91D8: 04 46 48 >128           db    dest,addr,AX,0
91DC: 1B 96    >129           da    CALLREQ
               >130
91DE: 42 4F 4F >131           asc   'BOOT',00                  ; &BOOT
91E3: 46 48 52 >132           db    addr,lngth,locadr,0
91E7: 8E 94    >133           da    BOOTREQ
               >134
91E9: 42 43 41 >135           asc   'BCAST',00                 ; &BCAST
91EF: 46 48 52 >136           db    addr,lngth,locadr,0
91F3: 97 94    >137           da    BCASTREQ
               >138
91F5: 42 B9 00 >139           db    'B',POKE_t,0               ; &BPOKE
91F8: 46 48 00 >140           db    addr,val,0
91FB: 01 96    >141           da    BPOKEREQ
               >142
91FD: 49 4E 49 >143           asc   'INIT',00                  ; &INIT
9202: 00       >144           db    0
9203: A8 93    >145           da    INIT
               >146
9205: 54 49 4D >147           asc   'TIMEOUT',00               ; &TIMEOUT
920D: 14 00    >148           db    n60ms,0
920F: 64 93    >149           da    timeout
               >150
9211: 49 44 54 >151           asc   'IDTBL',00                 ; &IDTBL
9217: D0 00    >152           db    val?,0
9219: 71 93    >153           da    idtbl
               >154
921B: 00       >155           db    0          ; End of Command Table
```

```
                >157  ************************************************************
                >158  *                                                          *
                >159  *                     I N S T A L L                        *
                >160  *                                                          *
                >161  *           Michael J. Mahon - Oct 25, 2004                *
                >162  *                Revised Aug 16, 2008                      *
                >163  *                                                          *
                >164  *              Copyright (c) 2004, 2008                    *
                >165  *                                                          *
                >166  *  Installs AmperNada as first ampersand routine (if not   *
                >167  *  installed already) and chains to an existing routine.   *
                >168  *  if no routine is currently installed, it defaults to    *
                >169  *  "SYNTAX ERROR".                                         *
                >170  *                                                          *
                >171  ************************************************************
                >172
921C: AD 7A 91  >173  INSTALL  lda   instald    ; AmperNada installed?
921F: D0 23     >174           bne   :exit      ; -Yes, don't repeat.
9221: A9 4C     >175           lda   #$4C       ; -No, set flag and install.
9223: 8D 7A 91  >176           sta   instald
9226: CD F5 03  >177           cmp   AMPVECT    ; Is "&" vector a JMP?
9229: 8D F5 03  >178           sta   AMPVECT    ; (always set "jmp")
922C: D0 0C     >179           bne   :setvect   ; -No, just set vector.
                >180  :chain   mov16 AMPVECT+1;chain+1 ; -Yes, chain to it.
922E: AD F6 03  >180           lda   AMPVECT+1  ; Move 2 bytes
9231: 8D 5E 92  >180           sta   chain+1
9234: AD F7 03  >180           lda   1+AMPVECT+1
9237: 8D 5F 92  >180           sta   1+chain+1
                >180           eom
                >181  :setvect mov16 #AMPNADA;AMPVECT+1 ; set the vector.
923A: A9 47     >181           lda   #AMPNADA   ; Move 2 bytes
923C: 8D F6 03  >181           sta   AMPVECT+1
923F: A9 92     >181           lda   #AMPNADA/$100 ; high byte of immediate
9241: 8D F7 03  >181           sta   1+AMPVECT+1
                >181           eom
9244: 4C A8 93  >182  :exit    jmp   INIT       ; Initialize NadaNet.
```

```
                >184  ************************************************************
                >185  *                                                          *
                >186  *                    A M P E R N A D A                     *
                >187  *                                                          *
                >188  *            Michael J. Mahon - Oct 25, 2004               *
                >189  *                 Revised Nov 08, 2004                     *
                >190  *                                                          *
                >191  *                  Copyright (c) 2004                      *
                >192  *                                                          *
                >193  *  Implements an ampersand (&) interface to NadaNet for    *
                >194  *  Applesoft programs.  Reduces the need for PEEKs and     *
                >195  *  POKEs to set up parameters, saving time and interface   *
                >196  *  definitions.                                            *
                >197  *                                                          *
                >198  ************************************************************
                >199
9247: 08        >200  AMPNADA  php             ; Save status
9248: 48        >201          pha             ;  and A for chain.
9249: A2 00     >202          ldx   #0
924B: 8E 7B 91  >203          stx   nparms    ; # of parms supplied
924E: 8E 7D 91  >204          stx   varcmd    ; Signal no var params seen
9251: 8E 7C 91  >205          stx   errstop   ; Clear "throw err" flag.
9254: A0 00     >206  cmd      ldy   #0        ; Start compare at TXTPTR
9256: BD 81 91  >207          lda   cmdtable,x ; Get command char
9259: D0 05     >208          bne   comp      ; -Not end, compare.
925B: 68        >209          pla             ; -End.  Restore A
925C: 28        >210          plp             ;   and status and chain
925D: 4C C9 DE  >211  chain    jmp   SYNERR    ;    to next & handler.
                >212
9260: D1 B8     >213  comp     cmp   (TXTPTR),y ; Does cmd match text?
9262: D0 09     >214          bne   :skipcmd  ; -No, skip this one.
9264: C8        >215          iny             ; -Yes, advance.
9265: E8        >216          inx
9266: BD 81 91  >217          lda   cmdtable,x ; End of command?
9269: D0 F5     >218          bne   comp      ; -No, keep comparing.
926B: F0 11     >219          beq   :doit     ; -Yes, go do it.
                >220
926D: E8        >221  :skipcmd inx             ; Skip to end of
926E: BD 81 91  >222          lda   cmdtable,x ;  current cmd string
9271: D0 FA     >223          bne   :skipcmd
9273: E8        >224  :skipp   inx             ; Skip to end of
9274: BD 81 91  >225          lda   cmdtable,x ;  current parm vect
9277: D0 FA     >226          bne   :skipp
9279: E8        >227          inx             ; Pass end mark
927A: E8        >228          inx             ;  and action
927B: E8        >229          inx             ;   routine address.
927C: D0 D6     >230          bne   cmd       ; Go check next command.
                >231
927E: 68        >232  :doit    pla             ; Discard entry A
927F: 68        >233          pla             ;  and status.
9280: B1 B8     >234          lda   (TXTPTR),y ; Look at next character.
9282: C8        >235          iny             ; (provisional match)
9283: C9 23     >236          cmp   #'#'      ; Is it "#"?
```

```
9285: F0 04    >237           beq   :advance   ; -Yes, don't throw error.
9287: 88       >238           dey              ; -No, don't match, and
9288: EE 7C 91 >239           inc   errstop    ;   set throw err flag.
928B: 20 98 D9 >240  :advance jsr   ADDON      ; Advance TXTPTR past cmd
928E: A9 28    >241           lda   #'('       ; Require initial "("
9290: 20 C0 DE >242  :nxparm  jsr   SYNCHR     ; Syntax err if no match.
9293: F0 61    >243           beq   :synerr    ; End not expected.
9295: 86 EC    >244           stx   cmdptr     ; Save for :done case
9297: C9 29    >245           cmp   #')'       ; Found a ")"?
9299: F0 5E    >246           beq   :done      ; -Yes, end of parm list.
929B: EE 7B 91 >247           inc   nparms     ; -No, another parm.
929E: E8       >248           inx              ; Advance ptr and
929F: BD 81 91 >249           lda   cmdtable,x ;  get parm descriptor.
92A2: F0 52    >250           beq   :synerr    ; Too many parms.
92A4: 85 ED    >251           sta   cmdsave    ; Save descriptor
92A6: 29 3F    >252           and   #$3F       ; Mask displacement
92A8: 85 EF    >253           sta   disp       ;  and save it.
92AA: 86 EC    >254           stx   cmdptr     ; Save pointer.
92AC: 24 ED    >255           bit   cmdsave    ; Test parm type.
92AE: 30 20    >256           bmi   :var       ; -Var parm
92B0: 50 12    >257           bvc   :byte      ; -Byte value parm
92B2: 20 67 DD >258           jsr   FRMNUM     ; -Word value parm
92B5: 20 52 E7 >259           jsr   GETADR     ; Word val to Y,A
92B8: A6 EF    >260           ldx   disp
92BA: 9D 39 91 >261           sta   @+1,x      ; Store the value
92BD: 98       >262           tya
92BE: 9D 38 91 >263           sta   @,x
92C1: 4C E7 92 >264           jmp   :more?
               >265
92C4: 20 F8 E6 >266  :byte    jsr   GETBYT     ; Byte value to X
92C7: A4 EF    >267           ldy   disp
92C9: 8A       >268           txa
92CA: 99 38 91 >269           sta   @,y        ; Store the value
92CD: 4C E7 92 >270           jmp   :more?
               >271
92D0: A5 ED    >272  :var     lda   cmdsave    ; Save the parm
92D2: 8D 7D 91 >273           sta   varcmd     ;  descriptor.
92D5: 20 E3 DF >274           jsr   PTRGET     ; Get var ptr in (A,Y)
92D8: 8D 7F 91 >275           sta   varadr     ;  and save var
92DB: 8C 80 91 >276           sty   varadr+1   ;   address.
92DE: A5 11    >277           lda   VALTYP     ; $FF if string
92E0: D0 14    >278           bne   :synerr    ; String not allowed.
92E2: A5 12    >279           lda   INTFLG     ; $80 if INT%
92E4: 8D 7E 91 >280           sta   vartype    ; Save for later use
92E7: 20 B7 00 >281  :more?   jsr   CHRGOT     ; Check current test char.
92EA: F0 0A    >282           beq   :synerr    ; End not expected.
92EC: C9 29    >283           cmp   #')'       ; Closing ")"?
92EE: F0 09    >284           beq   :done      ; -Yes, finish.
92F0: A6 EC    >285           ldx   cmdptr     ; -No, more parms.
92F2: A9 2C    >286           lda   #','       ; Require a comma.
92F4: D0 9A    >287           bne   :nxparm    ; (always)
               >288
92F6: 4C C9 DE >289  :synerr  jmp   SYNERR     ; SYNTAX ERROR
```

```
                >290
92F9: 20 B1 00  >291  :done    jsr   CHRGET     ; Pass the ")"
92FC: A6 EC     >292           ldx   cmdptr
92FE: E8        >293  :skipit  inx               ; Skip to end
92FF: BD 81 91  >294           lda   cmdtable,x ;  of parm descriptors.
9302: D0 FA     >295           bne   :skipit
                >296           mov16 cmdtable+1,x;:jsr+1 ; Action routine
9304: BD 82 91  >296           lda   cmdtable+1,x ; Move 2 bytes
9307: 8D 11 93  >296           sta   :jsr+1
930A: BD 83 91  >296           lda   1+cmdtable+1,x
930D: 8D 12 93  >296           sta   1+:jsr+1
                >296           eom
9310: 20 00 00  >297  :jsr     jsr   0*0         ; Call the action routine
9313: 85 00     >298           sta   $00         ; Save returned A
9315: A9 00     >299           lda   #0
9317: 2A        >300           rol               ; C to low bit
9318: 85 01     >301           sta   $01         ; Save returned Carry
931A: F0 0A     >302           beq   :noerr      ; No error, continue.
931C: AD 7C 91  >303           lda   errstop     ; Throw error?
931F: F0 0A     >304           beq   :rts        ; -No, just return.
9321: A2 31     >305           ldx   #49         ; -Yes, throw "DATA"
9323: 4C 12 D4  >306           jmp   ERROR       ;   error.
                >307
9326: AD 7D 91  >308  :noerr   lda   varcmd      ; Var parm passed?
9329: D0 01     >309           bne   :store      ; -Yes, store into it.
932B: 60        >310  :rts     rts               ; -No, return.
                >311
932C: 29 3F     >312  :store   and   #$3F        ; Mask displacement
932E: A8        >313           tay
932F: B9 38 91  >314           lda   @,y         ; Get low byte
9332: AA        >315           tax               ; X = lo byte of value
9333: A9 00     >316           lda   #0          ; Hi byte if byte value
9335: 2C 7D 91  >317           bit   varcmd      ; Is it byte or word?
9338: 50 03     >318           bvc   :byteval    ; -Byte, use 0 hi byte
933A: B9 39 91  >319           lda   @+1,y       ; -Word, get hi byte
933D: A8        >320  :byteval tay               ; Y = hi byte of value
                >321           mov16 varadr;FORPNT ; Address of variable
933E: AD 7F 91  >321           lda   varadr      ; Move 2 bytes
9341: 85 85     >321           sta   FORPNT
9343: AD 80 91  >321           lda   1+varadr
9346: 85 86     >321           sta   1+FORPNT
                >321           eom
9348: AD 7E 91  >322           lda   vartype     ; INT% or FLOAT variable?
934B: 10 0A     >323           bpl   :float      ; -FLOAT
934D: 98        >324           tya               ; -INT%
934E: A0 00     >325           ldy   #0          ; Store hi byte
9350: 91 85     >326           sta   (FORPNT),y ;  in INT% variable.
9352: C8        >327           iny               ; Point to lo byte
9353: 8A        >328           txa               ; Store lo byte
9354: 91 85     >329           sta   (FORPNT),y ;  in INT% variable.
9356: 60        >330           rts
                >331
9357: 84 9E     >332  :float   sty   FAC+1       ; Hi byte to FAC
```

```
9359: 86 9F    >333           stx   FAC+2      ; Lo byte to FAC
935B: A2 90    >334           ldx   #$90       ; Binary point 16 bits right
935D: 38       >335           sec             ; (Don't negate FAC)
935E: 20 A0 EB >336           jsr   FLO2       ; Normalize FAC
9361: 4C 27 EB >337           jmp   SETFOR     ; Pack FAC into variable.
```

```
                >339  ************************************************************
                >340  *                                                          *
                >341  *                  &TIMEOUT ([n60ms])                      *
                >342  *                                                          *
                >343  *           Michael J. Mahon - Oct 28, 2004                *
                >344  *                                                          *
                >345  *                  Copyright (c) 2004                      *
                >346  *                                                          *
                >347  *  Set new request timeout value in units of 60 ms.       *
                >348  *                                                          *
                >349  *  If no value is supplied, reset timeout to default.     *
                >350  *                                                          *
                >351  ************************************************************
                >352
9364: AD 7B 91  >353  timeout  lda   nparms     ; Parm supplied?
9367: D0 05     >354           bne   null       ; -Yes, timeout set.
9369: A9 32     >355           lda   #maxretry  ; -No, restore
936B: 8D 4C 91  >356           sta   retrylim   ;   the default.
936E: 68        >357  null     pla              ; No post-action
936F: 68        >358           pla              ;  processing needed.
9370: 60        >359           rts
```

```
                >361  ************************************************************
                >362  *                                                          *
                >363  *                      &IDTBL (val?)                       *
                >364  *                                                          *
                >365  *            Michael J. Mahon - Nov 05, 2004               *
                >366  *                                                          *
                >367  *                  Copyright (c) 2004                      *
                >368  *                                                          *
                >369  *  Return address of 'idtable' in parm variable.           *
                >370  *                                                          *
                >371  ************************************************************
                >372
                >373  idtbl    mov16 #idtable;rbuf+len ; Put addr in rbuf
9371: A9 5A     >373           lda   #idtable   ; Move 2 bytes
9373: 8D 48 91  >373           sta   rbuf+len
9376: A9 91     >373           lda   #idtable/$100 ; high byte of immediate
9378: 8D 49 91  >373           sta   1+rbuf+len
                >373           eom
937B: 18        >374           clc
937C: 60        >375           rts
```

```
                 57            put   INITSERVE
                >2     *** Table of service routines used by SERVER ***
                >3
937D: 12 95     >4     service  dw    PEEKSRV    ; Table of service routines
937F: A6 95     >5              dw    POKESRV    ; (Must be in order)
9381: 2E 96     >6              dw    CALLSRV
9383: 62 94     >11             dw    ]PROTERR   ; (Error if PUTMSG)
9385: 62 94     >12             dw    ]PROTERR   ; (Error if GETMSG)
9387: B2 94     >15             dw    GETIDSRV   ; Master serves GETID
9389: 62 94     >19             dw    ]PROTERR   ; (Error if non-bcast BOOT)
938B: 98 99     >20             dw    rarl=>al   ; (Data handled by caller)
938D: 0C 96     >21             dw    BPOKESRV
938F: 48 95     >22             dw    PKINCSRV
9391: 91 95     >26             dw    RUNSRV
9393: A6 95     >28             dw    BRUNSRV
                >29
                >30     * Version message printed by INIT
                >31
9395: CE C1 C4  >32     vermsg   asc   "NADANET "
939D: B3        >33              db    version/16."0" ; Major version #
939E: AE        >34              asc   "."
939F: B0        >35              db    version&$0F."0" ; Minor version #
93A0: AC A0 C9  >39              asc   ", ID = $"
                >40     verlen   equ   *-vermsg   ; Length of msg
```

```
                >42   ************************************************************
                >43   *                                                          *
                >44   *                        I N I T                           *
                >45   *                                                          *
                >46   *             Michael J. Mahon - Mar 5, 2004               *
                >47   *                  Revised May 21, 2008                    *
                >48   *                                                          *
                >49   *          Copyright (c) 1996, 2004, 2005, 2008            *
                >50   *                                                          *
                >51   *  Initialize NADANET, sign on, and return to caller.      *
                >52   *                                                          *
                >53   ************************************************************
                >54
93A8: AD CC 03  >55   INIT     lda   bootself   ; Set up ID from BOOT
93AB: 20 DA 93  >56            jsr   setid
93AE: B0 29     >57            bcs   :err       ; Bad ID, no INIT.
93B0: A9 4C     >58            lda   #$4C       ; Set warmstrt JMP to
93B2: 8D CD 03  >59            sta   warmstrt   ;  servlp (& nadapage)
                >60            mov16 #servelp;warmstrt+1
93B5: A9 03     >60            lda   #servelp   ; Move 2 bytes
93B7: 8D CE 03  >60            sta   warmstrt+1
93BA: A9 91     >60            lda   #servelp/$100 ; high byte of immediate
93BC: 8D CF 03  >60            sta   1+warmstrt+1
                >60            eom
93BF: 20 8B FD  >61            jsr   CROUT1     ; New line.
93C2: A0 00     >62            ldy   #0
93C4: B9 95 93  >63   :msgloop lda   vermsg,y   ; Print version message
93C7: 20 ED FD  >64            jsr   COUT
93CA: C8        >65            iny
93CB: C0 13     >66            cpy   #verlen
93CD: 90 F5     >67            bcc   :msgloop
93CF: AD 39 91  >68            lda   self       ;  and current ID.
93D2: 20 DA FD  >69            jsr   PRBYTE     ;   (in hex)
93D5: 20 8B FD  >70            jsr   CROUT1     ; New line.
93D8: 18        >71            clc              ; Good return.
93D9: 60        >72   :err     rts
```

```
                >75   ************************************************************
                >76   *                                                          *
                >77   *                        S E T I D                         *
                >78   *                                                          *
                >79   *              Michael J. Mahon - May 13, 2004             *
                >80   *                    Revised Aug 17, 2008                  *
                >81   *                                                          *
                >82   *                  Copyright (c) 2004, 2008                *
                >83   *                                                          *
                >84   *  Set machine ID to contents of A register and reset      *
                >85   *  the arbitration delay to 'arbtime' plus 22 cycles       *
                >86   *  times the machine ID, to avoid collisions.              *
                >87   *                                                          *
                >88   *  Delay from last arbitration poll to bus lock is 10      *
                >89   *  cycles, so 22 (2 * 11 cycles) increment provides a      *
                >90   *  little insurance.                                       *
                >91   *                                                          *
                >92   ************************************************************
                >93
93DA: 8D 39 91  >94   setid    sta   self       ; Machine ID
93DD: 8D 3D 91  >95            sta   sbuf+frm   ; Set sender field.
93E0: 49 FF     >96            eor   #$FF       ; Complement ID
93E2: 8D 3B 91  >97            sta   sbuf+frmc  ;  for collision detect.
93E5: 49 FF     >98            eor   #$FF       ; Back to ID
93E7: 38        >99            sec              ; Anticipate error.
93E8: F0 09     >100           beq   :err       ; -Error if zero.
93EA: 18        >101           clc              ; Anticipate no error.
93EB: 30 03     >102           bmi   :setarb    ; -Use temp ID (>127)
93ED: 0A        >103           asl              ; Mult ID by 2
93EE: 69 5C     >104           adc   #arbx      ;  and add to base
93F0: 8D 4E 91  >105  :setarb  sta   arbxv      ;   arb delay.
93F3: 60        >106  :err     rts
```

```
                        >109  ********************************************************
                        >110  *                                                      *
                        >111  *                      S E R V E R                     *
                        >112  *                                                      *
                        >113  *             Michael J. Mahon - May 5, 1996           *
                        >114  *                  Revised Oct 06, 2008                *
                        >115  *                                                      *
                        >116  *             Copyright (c) 1996, 2004, 2008           *
                        >117  *                                                      *
                        >118  *  SERVER continually listens to the net, receiving all  *
                        >119  *  packets, and responding to control packets directed   *
                        >120  *  to 'self'.  If a key is pressed or a request handled, *
                        >121  *  SERVER returns. C = 0 if count expired and is set as  *
                        >122  *  the request server left it if a request was handled.  *
                        >123  *                                                      *
                        >124  *  To minimize missed polls, SERVER temporarily raises   *
                        >125  *  RCVPKT's timeout to 20ms. from the normal value equal *
                        >126  *  to the minimum arbitration time.                     *
                        >127  *                                                      *
                        >128  *  For every request code, there is a corresponding     *
                        >129  *  server routine.  SERVER invokes these routines to    *
                        >130  *  satisfy the service requests it receives.  Upon entry *
                        >131  *  to 'xxxSRV', C = 0 and (X) = (rbuf+rqmd).            *
                        >132  *                                                      *
                        >133  *  To ensure that the next packet received is the start  *
                        >134  *  packet of a request protocol, it is necessary to wait *
                        >135  *  for the net to be idle or locked for at least the min *
                        >136  *  arbitration time before receiving a request.  (Note   *
                        >137  *  that broadcast requests begin with the network in a   *
                        >138  *  locked state.)                                       *
                        >139  *                                                      *
                        >140  *  The entry point 'svrxkbd' is provided for 'servelp',  *
                        >141  *  which ignores keyboard input.                        *
                        >142  *                                                      *
                        >143  ********************************************************
                        >144
93F4: AD 10 C0          >145  svrxkbd  lda   kbstrobe   ; Ignore any keypress
                        >146
93F7: A9 08             >147  SERVER   lda   #idleto    ; While polling, raise
93F9: 8D 4F 91          >148           sta   tolim      ;  RCVPKT timeout to 20ms.
93FC: A2 3A             >149  :resync  ldx   #servegap  ; Delay min arb time
93FE: CD E8 C0          >150           cmp   zipslow    ; Zip Chip to 1MHz mode.
9401: AC 62 C0          >151           ldy   drecv      ; Sample net state.
9404: 98                >152  :waitidl tya
9405: 4D 62 C0          >153           eor   drecv      ; Has net changed?
9408: 30 F2             >154           bmi   :resync    ; -Yes, restart timing.
940A: CA                >155           dex              ; -No, count it down.
940B: D0 F7             >156           bne   :waitidl   ; -Keep waiting.
940D: AD 00 C0          >157  :serve   lda   keybd      ; Check if key pressed.
9410: 30 75             >158           bmi   :exit      ; -Yes, return.
9412: 20 00 99          >159           jsr   RCVCTL     ; Receive ctl pkt to 'rbuf'
9415: B0 69             >160           bcs   :err       ; -Timeout or Cksum err.
9417: AD 43 91          >161           lda   rbuf+frmc  ; -Cksum OK, verify that
```

```
941A: 49 FF    >162           eor   #$FF       ;   complement of 'frmc'
941C: CD 45 91 >163           cmp   rbuf+frm   ;    is equal to 'frm'.
941F: D0 55    >164           bne   :frmcerr   ; -No, count collisions.
9421: AD 44 91 >165           lda   rbuf+dst   ; -Yes, good packet.
9424: F0 2D    >166           beq   :bcastck   ; Broadcast packet OK?
9426: CD 39 91 >167           cmp   self       ; Directed to us?
9429: D0 3A    >168           bne   :skip      ; -No, just keep time.
942B: AD 42 91 >169  :bcast   lda   rbuf+rqmd  ; -Yes, get 'rqmd'
942E: AA       >170           tax              ;   and save in X.
942F: 29 07    >171           and   #modmask   ; Is the modifier
9431: C9 01    >172           cmp   #rm_REQ    ;  a Request?
9433: D0 2D    >173           bne   ]PROTERR   ; -No, protocol error.
9435: 8A       >174           txa              ; -Yes, check request.
9436: 29 F8    >175           and   #reqmask
9438: F0 28    >176           beq   ]PROTERR   ; Code must be > 0
943A: C9 68    >177           cmp   #maxreq    ;  and < maxreq.
943C: B0 24    >178           bcs   ]PROTERR   ; Invalid request.
943E: 4A       >179           lsr              ; Req code is * 8,
943F: 4A       >180           lsr              ;  so divide by 4. (C=0)
9440: A8       >181           tay              ; Index of service routine
               >182           mov16 service-2,y;address ; Set up address
9441: B9 7B 93 >182           lda   service-2,y ; Move 2 bytes
9444: 85 FC    >182           sta   address
9446: B9 7C 93 >182           lda   1+service-2,y
9449: 85 FD    >182           sta   1+address
               >182           eom
944B: A9 01    >183           lda   #reqto     ; Reset timeout to min
944D: 8D 4F 91 >184           sta   tolim      ;  arbitration time.
9450: 6C FC 00 >185           jmp   (address)  ; Jump to service routine.
               >186
9453: AD 42 91 >187  :bcastck lda   rbuf+rqmd  ; Ck broadcast valid..
9456: C9 49    >188           cmp   #r_BPOKE+rm_REQ ; BPOKE request?
9458: F0 D1    >189           beq   :bcast     ; -Yes, process request.
945A: C9 41    >190           cmp   #r_BCAST+rm_REQ ; Broadcast BCAST req?
945C: F0 CD    >191           beq   :bcast     ; -Yes.
945E: C9 39    >192           cmp   #r_BOOT+rm_REQ ; Broadcast BOOT req?
9460: F0 03    >193           beq   :skip      ; -Yes, ignore.
9462: 20 8F 99 >194  ]PROTERR jsr   PROTERR    ; Record protocol error
9465: CE 50 91 >195  :skip    dec   reqctr     ; Enough requests seen?
9468: D0 92    >196           bne   :resync    ; -No, re-sync SERVER.
946A: A9 03    >197           lda   #reqpidle  ; -Yes, about 20ms used.
946C: 8D 50 91 >198           sta   reqctr     ; Reset counter.
946F: CE 4D 91 >199           dec   servecnt   ; Enough iterations?
9472: F0 13    >200           beq   :exit      ; -Yes, return.
9474: D0 86    >201           bne   :resync    ; -No, re-sync SERVER.
               >202
               >203  :frmcerr inc16 frmcerr    ; Count sync'd collisions.
9476: EE 57 91 >203           inc   frmcerr    ; Increment 16-bit word.
9479: D0 03    >203           bne   *+5        ; - No carry.
947B: EE 58 91 >203           inc   frmcerr+1  ; Propagate carry.
               >203           eom
947E: D0 E5    >204           bne   :skip      ; (always)
               >205
```

```
9480: D0 E3    >206  :err     bne   :skip      ; -Cksum error.
9482: CE 4D 91 >207           dec   servecnt   ; -Timeout.  Enough?
9485: D0 86    >208           bne   :serve     ; -No, keep serving.
9487: A9 01    >209  :exit    lda   #reqto     ; -Yes, restore normal
9489: 8D 4F 91 >210           sta   tolim      ;   request timeout,
948C: 18       >211           clc             ;    clear Carry
948D: 60       >212           rts             ;     and return.
```

```
>216  *-------------------------------------------------------*
>217  *            Broadcast Boot & Bcast Protocol            *
>218  *-------------------------------------------------------*
>219  *          Master                      Slaves           *
>220  *  ======================     ======================   *
>221  *  Bxxx   REQ (addr,leng)  ====>                        *
>222  *                   (800 cyc. delay)                    *
>223  *                     Data  ====>                       *
>224  *                           :                           *
>225  *                     Data  ====>                       *
>226  *-------------------------------------------------------*
```

```
                >228  ********************************************************
                >229  *                                                      *
                >230  *          B O O T R E Q   &   B C A S T R E Q         *
                >231  *                                                      *
                >232  *            Michael J. Mahon - May 14, 2004           *
                >233  *                  Revised Oct 06, 2008                *
                >234  *                                                      *
                >235  *                Copyright (c) 2004, 2008              *
                >236  *                                                      *
                >237  *  Broadcast request for all waiting machines to receive *
                >238  *  data of 'sbuf+len' length.                          *
                >239  *                                                      *
                >240  *  BOOTREQ is handled by all machines awaiting boot.    *
                >241  *  The boot image following is loaded at 'sbuf+adr' and  *
                >242  *  control is passed to the boot image.                 *
                >243  *                                                      *
                >244  *  BCASTREQ is handled by all machines awaiting BCAST   *
                >245  *  data.  'sbuf+adr' is the "tag" for the data following,*
                >246  *  that is ignored or received by waiting machines based *
                >247  *  on their state and the tag value.                   *
                >248  *                                                      *
                >249  *  Since these requests are broadcast, they do not get  *
                >250  *  ACKs from their destination(s), but simply send their *
                >251  *  data blindly.  If errors occur, waiting machines will *
                >252  *  continue to wait for good data, so verification of   *
                >253  *  proper operation must be handled separately.         *
                >254  *                                                      *
                >255  *  Because broadcast data is sent "open loop", and since *
                >256  *  BCAST clients may require time to determine whether  *
                >257  *  and how they should receive the following data, these *
                >258  *  protocols delay for 800 cycles between the request   *
                >259  *  and the sending of data.                             *
                >260  *                                                      *
                >261  *  BOOTREQ & BCASTREQ do the following steps:           *
                >262  *     1. Sets up the request                           *
                >263  *     2. Does a broadcast arbitration to seize the net *
                >264  *        and delay 20ms to resolve any collisions and  *
                >265  *        allow slow pollers to reach their RCVPKT holds *
                >266  *     3. Sends the request, with address/tag and length *
                >267  *     4. Waits 800 cyc. for clients to prepare to      *
                >268  *        receive the data (or not).                    *
                >269  *     5. Sends the boot code/data stream               *
                >270  *                                                      *
                >271  ********************************************************
                >272
948E: A9 01     >273  BOOTREQ  lda   #rm_REQ    ; Put modifier code         #
9490: 8D 3B 91  >274           sta   sbuf+frmc  ;  into old 'mod' byte.     #
                >275  * Version 3 code (incompatible with v2.1 boot ROMs)
                >276  *        lda   #r_BOOT+rm_REQ
                >277  * v2.1 boot ROM compatibility patch                   #
                >278  * +----+----+----+----+----+----+----+----+          #
                >279  * |rqmd|frmc|dst |frm | address | length  | v3 ctl pkt#
                >280  * +----+----+----+----+----+----+----+----+          #
```

```
                >281  * +----+----+----+----+----+----+----+----+           #
                >282  * |req |mod |dst |frm | address | length  | v2 ctl pkt#
                >283  * +----+----+----+----+----+----+----+----+           #
                >284  *                                                      #
9493: A9 07     >285           lda   #r_BOOT/reqfac ; Unshift BOOT req code #
9495: D0 02     >286           bne   ]doit      ; (always)
                >287
9497: A9 41     >288  BCASTREQ lda   #r_BCAST+rm_REQ ; BCAST request
9499: 8D 3A 91  >289  ]doit    sta   sbuf+rqmd
949C: 20 12 97  >290           jsr   BCASTARB   ; Bcast arbitrate & lock bus
949F: 20 26 98  >291           jsr   SENDCTL    ; Send the BOOT request.
94A2: AD 3D 91  >292           lda   sbuf+frm   ; Restore 'frmc' field.     #
94A5: 49 FF     >293           eor   #$FF                                   #
94A7: 8D 3B 91  >294           sta   sbuf+frmc                              #
94AA: 20 E9 98  >295           jsr   lasl=>al   ; Local start address & length
94AD: A2 A0     >296           ldx   #800/5     ; Delay 800 cycles,
94AF: 4C AD 99  >297           jmp   DSENDLNG   ;  send data and return.
```

```
                >301  ************************************************************
                >302  *                                                          *
                >303  *                     G E T I D S R V                      *
                >304  *                                                          *
                >305  *            Michael J. Mahon - May 14, 2004               *
                >306  *                 Revised Aug 17, 2008                     *
                >307  *                                                          *
                >308  *                Copyright (c) 2004, 2008                  *
                >309  *                                                          *
                >310  *  Service machine 'rbuf+frm's request to allocate a new   *
                >311  *  machine ID (if 'rbuf+frm' is a pseudo-ID).              *
                >312  *                                                          *
                >313  *  The new ID is sent in the ACK packet.  GETIDSRV         *
                >314  *  requires a DACK packet from the newly allocated         *
                >315  *  machine ID before the new allocation is committed.      *
                >316  *                                                          *
                >317  *  GETIDSRV is unique in that it returns control           *
                >318  *  directly to SERVER (rather than SERVER's caller), so     *
                >319  *  that all GETID requests are processed before SERVER     *
                >320  *  returns.                                                *
                >321  *                                                          *
                >322  *  GETIDSRV does the following steps:                      *
                >323  *     1. If a pseudo-ID was received, it finds the next    *
                >324  *        available machine ID.                             *
                >325  *     2. Sends the new (or current) ID in the ACK packet   *
                >326  *     3. Receives the DACK and marks the ID allocated.     *
                >327  *     4. Gives control back to SERVER.                     *
                >328  *                                                          *
                >329  ************************************************************
                >330
94B2: AE 45 91  >331  GETIDSRV ldx   rbuf+frm   ; Look at requester's ID
94B5: 10 0E     >332           bpl   :ack       ; -it's real, just ACK.
94B7: A2 02     >333           ldx   #2         ; -pseudo, find new one.
94B9: BD 5A 91  >334  :search  lda   idtable,x  ; Find lowest
94BC: F0 07     >335           beq   :ack       ;  unused ID.
94BE: E8        >336           inx
94BF: E0 20     >337           cpx   #maxid+1
94C1: 90 F6     >338           bcc   :search
94C3: B0 19     >339           bcs   :exit      ; Table overflow!
                >340
94C5: 8E 3E 91  >341  :ack     stx   sbuf+adr   ; Send ID to requester
94C8: 20 12 98  >342           jsr   SENDACK
94CB: AD 3E 91  >343           lda   sbuf+adr   ; Expect new ID
94CE: 8D 3C 91  >344           sta   sbuf+dst   ;  in DACK.
94D1: 20 96 96  >345           jsr   RCVDACK
94D4: B0 08     >346           bcs   :exit      ; -Error, don't allocate.
94D6: AE 3C 91  >347           ldx   sbuf+dst   ; -OK.
94D9: A9 01     >348           lda   #1
94DB: 9D 5A 91  >349           sta   idtable,x  ; Allocate the ID
94DE: 4C F7 93  >350  :exit    jmp   SERVER     ; Go back to SERVER.
```

```
 58           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
94E1: A9 03             >37   PEEKREQ  lda   #maxreqrt  ; Set request retry
94E3: 8D 51 91          >38           sta   reqretry   ;  counter.
94E6: A9 08             >39   :retry   lda   #r_PEEK    ; Send PEEK request.
94E8: 20 3A 96          >40           jsr   REQUEST
94EB: B0 1E             >41           bcs   :failed
94ED: 20 E9 98          >42           jsr   lasl=>al   ; Set up address/length
94F0: A5 FF             >43           lda   length+1   ; If length
94F2: D0 12             >44           bne   :long      ;  is >255 bytes, or
94F4: A4 FE             >45           ldy   length     ;    if length is
94F6: F0 19             >46           beq   :done      ; (length = 0!)
94F8: C0 05             >47           cpy   #5         ;      > 4 bytes,
94FA: B0 0A             >48           bcs   :long      ;   receive multiple pkts.
94FC: 88                >49           dey              ; Move short response
94FD: B9 46 91          >50   :short   lda   rbuf+adr,y ;  to local data address.
9500: 91 FC             >51           sta   (address),y
9502: 88                >52           dey
9503: 10 F8             >53           bpl   :short
9505: 60                >54           rts              ; ...and return.
                        >55
9506: 20 CF 99          >56   :long    jsr   RCVLONG    ; Receive multiple packets
9509: 90 06             >57           bcc   :done      ; No problem.
950B: CE 51 91          >58   :failed  dec   reqretry   ; Dec request retry count
950E: D0 D6             >59           bne   :retry     ; Try until OK or exhausted,
9510: 38                >60           sec              ;  then return with C set.
9511: 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
9512: 20 98 99  >84   PEEKSRV  jsr   rarl=>al   ; Set address/length.
9515: A5 FF     >85            lda   length+1   ; Check for long response
9517: D0 14     >86            bne   :long
9519: A4 FE     >87            ldy   length     ; Check for < 5 bytes.
951B: F0 0D     >88            beq   :nullreq   ; length = 0.
951D: C0 05     >89            cpy   #5
951F: B0 0C     >90            bcs   :long      ; - No, longer.
9521: 88        >91            dey              ; - Yes, move response
9522: B1 FC     >92   :short   lda   (address),y ;   data into ACK packet.
9524: 99 3E 91  >93            sta   sbuf+adr,y
9527: 88        >94            dey
9528: 10 F8     >95            bpl   :short
952A: 4C 12 98  >96   :nullreq jmp   SENDACK    ; Send ACK with response.
                >97
952D: 20 12 98  >98   :long    jsr   SENDACK    ; ACK the request.
9530: A2 1C     >99            ldx   #140/5     ; Allow requester to receive.
9532: 4C AD 99 >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                    *
                >114  *                                                       *
                >115  *             Michael J. Mahon - Nov 05, 2004           *
                >116  *                                                       *
                >117  *                   Copyright (c) 2004                  *
                >118  *                                                       *
                >119  *  Request machine 'sbuf+dst' to return 2 bytes at its  *
                >120  *  'sbuf+adr', then increment that value by 'sbuf+len'. *
                >121  *  Put the returned, unincremented value at 'locaddr'.  *
                >122  *                                                       *
                >123  *  PEEKREQ, like other requests, will retry the request *
                >124  *  in case of error, up to 'maxreqrt' times.  If errors *
                >125  *  persist, it will return with Carry set.              *
                >126  *                                                       *
                >127  *  PEEKREQ does the following steps:                    *
                >128  *     1. Make the PEEKINC request (and receive the ACK) *
                >129  *     2. Move 2 bytes in 'sbuf+len' into 'locaddr'      *
                >130  *     3. Retry in case of error up to 'maxreqrt' times  *
                >131  *                                                       *
                >132  *********************************************************
                >133
9535: A9 03     >134  PKINCREQ lda   #maxreqrt  ; Set request retry
9537: 8D 51 91  >135           sta   reqretry   ;  counter.
953A: A9 50     >136  :retry   lda   #r_PKINC   ; Send PEEKINC request.
953C: 20 3A 96  >137           jsr   REQUEST
953F: 90 06     >138           bcc   :done      ; Done if no error.
9541: CE 51 91  >139           dec   reqretry   ; Dec request retry count
9544: D0 F4     >140           bne   :retry     ; Try until OK or exhausted,
9546: 38        >141           sec              ;  then return with C set.
9547: 60        >142  :done    rts
```

```
                >146  ************************************************************
                >147  *                                                          *
                >148  *                     P K I N C S R V                      *
                >149  *                                                          *
                >150  *             Michael J. Mahon - Nov 05, 2004              *
                >151  *                  Revised May 21, 2008                    *
                >152  *                                                          *
                >153  *                Copyright (c) 2004, 2008                  *
                >154  *                                                          *
                >155  *  Service machine 'rbuf+frm's request to send 2 bytes     *
                >156  *  at our 'rbuf+adr', then increment value by 'rbuf+len'.  *
                >157  *                                                          *
                >158  *  The PEEKINC request serves as a "network atomic"        *
                >159  *  read-modify-write primitive for synchronization and     *
                >160  *  allocation operations.                                  *
                >161  *                                                          *
                >162  *  PKINCSRV does the following steps:                      *
                >163  *     1. Save initial 2-byte value in ACK buffer           *
                >164  *        while incrementing the value by 'rbuf+len'        *
                >165  *     2. Send the ACK packet with data.                    *
                >166  *                                                          *
                >167  ************************************************************
                >168
9548: 20 A2 99  >169  PKINCSRV jsr   ra=>a      ; Set up data address
954B: A0 00     >170           ldy   #0
954D: 18        >171           clc
954E: B1 FC     >172  :movinc  lda   (address),y ; Move and Inc 2 bytes
9550: 99 40 91  >173           sta   sbuf+len,y
9553: 79 48 91  >174           adc   rbuf+len,y
9556: 91 FC     >175           sta   (address),y
9558: C8        >176           iny
9559: 98        >177           tya             ; Don't disturb carry.
955A: 49 02     >178           eor   #2        ; Done?
955C: D0 F0     >179           bne   :movinc   ; -No, go again.
955E: 4C 12 98  >180           jmp   SENDACK   ; -Yes, send ACK with value.
```

```
>182  *-------------------------------------------------------*
>183  *              POKE, RUN, and BRUN Protocol             *
>184  *-------------------------------------------------------*
>185  *         Requester                    Server           *
>186  *  ======================     ======================  *
>187  *  xxxx   REQ (addr,leng)  ====>                       *
>188  *                          <==== POKE   ACK            *
>189  *                    Data  ====>                       *
>190  *                            :                         *
>191  *                    Data  ====>                       *
>192  *                          <==== POKE   DACK           *
>193  *-------------------------------------------------------*
```

```
                        >197  ************************************************************
                        >198  *                                                          *
                        >199  *     P O K E R E Q ,  R U N R E Q ,   B R U N R E Q      *
                        >200  *                                                          *
                        >201  *            Michael J. Mahon - May 11, 1996               *
                        >202  *                  Revised Sep 25, 2008                    *
                        >203  *                                                          *
                        >204  *             Copyright (c) 1996, 2004, 2008               *
                        >205  *                                                          *
                        >206  *  Request machine 'sbuf+dst' to store 'sbuf+len' bytes    *
                        >207  *  at its 'sbuf+adr', and send them from our location      *
                        >208  *  'locaddr'.                                              *
                        >209  *                                                          *
                        >210  *  These requests, like others, will retry the request     *
                        >211  *  in case of error, up to 'maxreqrt' times.  If errors    *
                        >212  *  persist, it will return with Carry set.                 *
                        >213  *                                                          *
                        >214  *  POKEREQ, RUNREQ, and BRUNREQ do the following steps:    *
                        >215  *     1. Make the request (and receive the ACK)            *
                        >216  *     2. Send 'sbuf+len' bytes of data from 'locaddr'      *
                        >217  *     3. Receive DATA ACK response                         *
                        >218  *     4. Retry in case of error up to 'maxreqrt' times     *
                        >219  *                                                          *
                        >220  ************************************************************
                        >221
9561: A9 58             >222  RUNREQ   lda   #r_RUN     ; Send RUN request.
9563: D0 06             >223          bne   setreq     ; (always)
                        >224
9565: A9 60             >225  BRUNREQ  lda   #r_BRUN    ; Send BRUN request.
9567: D0 02             >226          bne   setreq     ; (always)
                        >227
9569: A9 10             >228  POKEREQ  lda   #r_POKE    ; Send POKE request.
956B: 8D 3A 91          >229  setreq   sta   sbuf+rqmd  ; Set request code
956E: A2 03             >230          ldx   #maxreqrt  ; Set request retry
9570: 8E 51 91          >231          stx   reqretry   ;  counter.
9573: AD 3A 91          >232  :retry   lda   sbuf+rqmd  ; Recover request code
9576: 20 3A 96          >233          jsr   REQUEST
9579: B0 0B             >234          bcs   :failed
957B: 20 E9 98          >235          jsr   lasl=>al   ; Set up address/length.
957E: 20 B0 99          >236          jsr   SENDLONG   ; Send multiple packets.
9581: 20 96 96          >237          jsr   RCVDACK    ; Receive DATA ACK packet.
9584: 90 06             >238          bcc   :done      ; -OK, return.
9586: CE 51 91          >239  :failed  dec   reqretry   ; Dec request retry count
9589: D0 E8             >240          bne   :retry     ; Try until OK or exhausted,
958B: 38                >241          sec              ;  then return with C set.
958C: 60                >242  :done    rts
```

```
                        >246  ************************************************************
                        >247  *                                                          *
                        >248  *    P O K E S R V ,   R U N S R V ,   B R U N S R V       *
                        >249  *                                                          *
                        >250  *             Michael J. Mahon - May 11, 1996              *
                        >251  *                   Revised Jan 24, 2009                   *
                        >252  *                                                          *
                        >253  *              Copyright (c) 1996, 2008, 2009              *
                        >254  *                                                          *
                        >255  *  Service machine 'rbuf+frm's request to poke 'rbuf+len'* *
                        >256  *  bytes of data to our 'rbuf+adr'.                        *
                        >257  *                                                          *
                        >258  *  If RUNSRV, initialize Applesoft and RUN the BASIC      *
                        >259  *  program transferred.  (Address must be > $800.)         *
                        >260  *                                                          *
                        >261  *  If BRUNSRV, CALL the code transferred with (A,X) set   *
                        >262  *  to the code's load address.                             *
                        >263  *                                                          *
                        >264  *  POKESRV, RUNSRV, and BRUNSRV do the following steps:   *
                        >265  *     1. If RUNSRV: lock net, save CSW/KSW hooks, and      *
                        >266  *        coldstart Applesoft.                              *
                        >267  *     2. Send the ACK packet                               *
                        >268  *     3. Receive multiple packets to 'rbuf+adr'            *
                        >269  *     4. If data received OK, send DATA ACK packet         *
                        >270  *     5. If BRUNSRV: Set (A,X) to address & JMP to code.  *
                        >271  *     6. If RUNSRV: Init Applesoft ptrs, fix up links,     *
                        >272  *        restore CSW/KSW hooks, and RUN the program.       *
                        >273  *                                                          *
                        >274  ************************************************************
                        >275
                        >277  savhooks equ   $2FC       ; Save hooks at end of page 2
958D: 00 96             >278  sethooks dw    rts        ; For COLDSTRT and FIXLINKS
958F: A6 95             >279           dw    POKESRV    ;  use hooks to retain control.
                        >280
9591: 8D 5B C0          >281  RUNSRV   sta   dsend+1    ; Lock net for coldstart
9594: A2 03             >282           ldx   #3          ; Save and set CSW/KSW hooks
9596: B5 36             >283  :saveset lda   CSW,x       ;  to <rts,POKESRV> to retain
9598: 9D FC 02          >284           sta   savhooks,x ;   control after coldstart.
959B: BD 8D 95          >285           lda   sethooks,x
959E: 95 36             >286           sta   CSW,x
95A0: CA                >287           dex
95A1: 10 F3             >288           bpl   :saveset
95A3: 4C 00 E0          >289           jmp   COLDSTRT   ;    BASIC coldstart.
                        >291  BRUNSRV  equ   *
95A6: 20 12 98          >292  POKESRV  jsr   SENDACK    ; ACK the request.
95A9: 20 98 99          >293           jsr   rarl=>al   ; Set up address/length.
95AC: 20 CF 99          >294           jsr   RCVLONG    ; Receive long data message.
95AF: B0 4F             >295           bcs   rts        ; Receive error.
                        >296           delay 40         ; Allow requester to receive.
95B1: A2 08             >296           ldx   #40/5      ; (5 cycles per iteration)
95B3: CA                >296  ]delay   dex
95B4: D0 FD             >296           bne   ]delay
                        >296           eom
```

```
95B6: A9 03    >297           lda   #rm_DACK   ; Send DATA ACK
95B8: 20 14 98 >298           jsr   SENDRSP    ;   packet.
95BB: AD 42 91 >299           lda   rbuf+rqmd  ; Recover request
95BE: C9 11    >300           cmp   #r_POKE+rm_REQ ; POKE?
95C0: F0 3D    >301           beq   :ok        ; -Yes, return.
95C2: C9 61    >302           cmp   #r_BRUN+rm_REQ ; -No, BRUN?
95C4: F0 6B    >303           beq   docall     ; -Yes, do CALL.
95C6: AD CF 03 >307           lda   nadapage   ; -No, RUN.  Set HIMEM to
95C9: 85 74    >308           sta   HIMEM+1    ;   NadaNet load page.
95CB: 85 70    >309           sta   FRETOP+1
95CD: 18       >310           clc
95CE: AD 46 91 >311           lda   rbuf+adr   ; Set PSTART to start
95D1: 85 67    >312           sta   PSTART     ;  addr and VARTAB to
95D3: 6D 48 91 >313           adc   rbuf+len   ;   end of program.
95D6: 85 69    >314           sta   VARTAB
95D8: AD 47 91 >315           lda   rbuf+adr+1
95DB: 85 68    >316           sta   PSTART+1
95DD: 6D 49 91 >317           adc   rbuf+len+1
95E0: 85 6A    >318           sta   VARTAB+1
               >319           mov16 #:run;KSW  ; Retain control after
95E2: A9 ED    >319           lda   #:run      ; Move 2 bytes
95E4: 85 38    >319           sta   KSW
95E6: A9 95    >319           lda   #:run/$100 ; high byte of immediate
95E8: 85 39    >319           sta   1+KSW
               >319           eom
95EA: 4C F2 D4 >320           jmp   FIXLINKS   ;  fixing up prog links.
95ED: A2 04    >321  :run     ldx   #4         ; Restore CSW/KSW hooks.
95EF: BD FB 02 >322  :restore lda   savhooks-1,x
95F2: 95 35    >323           sta   CSW-1,x
95F4: CA       >324           dex
95F5: D0 F8    >325           bne   :restore
95F7: 8A       >326           txa              ; Set byte preceding
95F8: 81 B8    >327           sta   (TXTPTR,x) ;  program to zero,
95FA: 85 D8    >328           sta   ONERR      ;   clear ONERR flag, and
95FC: 4C 66 D5 >329           jmp   RUNPROG    ; RUN the Applesoft prog.
               >331
95FF: 18       >332  :ok      clc              ; Good return.
9600: 60       >333  rts      rts              ; Return.
```

```
                >335  *-----------------------------------------------------*
                >336  *         Requester                    Server         *
                >337  *  ======================     ======================  *
                >338  *  BPOKE  REQ (addr,val)   ====>                      *
                >339  *                                (Broadcast, No ACK)   *
                >340  *-----------------------------------------------------*
                >341
                >344  *******************************************************
                >345  *                                                     *
                >346  *                    B P O K E R E Q                  *
                >347  *                                                     *
                >348  *            Michael J. Mahon - Nov. 04, 2004         *
                >349  *                  Revised Aug 20, 2008               *
                >350  *                                                     *
                >351  *               Copyright (c) 2004, 2008              *
                >352  *                                                     *
                >353  *  Broadcast request to all serving machines to store 2  *
                >354  *  bytes in 'sbuf+len' at address 'sbuf+adr'.         *
                >355  *                                                     *
                >356  *  BPOKE, unlike most requests, is broadcast, and so  *
                >357  *  is not acknowledged by any receiver.  To eliminate  *
                >358  *  the chance of collision, it holds the bus locked for  *
                >359  *  20ms after arbitration, then sends the request packet.*
                >360  *  This allows enough time for any colliding sender to  *
                >361  *  send its request and re-arbitrate while the bus is  *
                >362  *  locked, so that there is no contention when the BPOKE *
                >363  *  request is finally sent.                           *
                >364  *                                                     *
                >365  *  BPOKEREQ does the following steps:                 *
                >366  *     1. Broadcast arbitrate and lock the bus         *
                >367  *     2. Set up BPOKE request                         *
                >368  *     3. Send the BPOKE request packet.               *
                >369  *                                                     *
                >370  *******************************************************
                >371
9601: 20 12 97  >372  BPOKEREQ jsr   BCASTARB   ; Bcast arbitrate & lock bus
9604: A9 49     >373          lda   #r_BPOKE+rm_REQ ; Set up BPOKE request.
9606: 8D 3A 91  >374          sta   sbuf+rqmd
9609: 4C 26 98  >375          jmp   SENDCTL    ; Send the request.
```

```
                >379  ************************************************************
                >380  *                                                          *
                >381  *                     B P O K E S R V                      *
                >382  *                                                          *
                >383  *             Michael J. Mahon - Nov 05, 2004              *
                >384  *                  Revised May 21, 2008                    *
                >385  *                                                          *
                >386  *                Copyright (c) 2004, 2008                  *
                >387  *                                                          *
                >388  *  Service machine 'rbuf+frm's request to poke 2 bytes     *
                >389  *  of data in 'rbuf+len' to our 'rbuf+adr'.                *
                >390  *                                                          *
                >391  *  BPOKESRV does the following:                            *
                >392  *     1. Move 'rbuf+len' to memory at 'rbuf+adr'.          *
                >393  *                                                          *
                >394  ************************************************************
                >395
960C: 20 A2 99  >396  BPOKESRV jsr   ra=>a      ; Set up pointer
960F: A0 01     >397           ldy   #1         ; and move 2 bytes.
9611: B9 48 91  >398  :move    lda   rbuf+len,y
9614: 91 FC     >399           sta   (address),y
9616: 88        >400           dey
9617: 10 F8     >401           bpl   :move
9619: 18        >402           clc
961A: 60        >403           rts              ; All done.
```

```
                >405  *-------------------------------------------------------*
                >406  *         Requester                    Server           *
                >407  *  ========================    ======================   *
                >408  *  CALL   REQ (addr,A,X)   ====>                        *
                >409  *                          <==== CALL   ACK             *
                >410  *-------------------------------------------------------*
                >411
                >414  *********************************************************
                >415  *                                                       *
                >416  *                    C A L L R E Q                      *
                >417  *                                                       *
                >418  *           Michael J. Mahon - May 11, 1996             *
                >419  *                Revised March 05, 2004                 *
                >420  *                                                       *
                >421  *              Copyright (c) 1996, 2004                 *
                >422  *                                                       *
                >423  *  Request machine 'sbuf+dst' to call a subroutine at    *
                >424  *  address 'sbuf+adr' with parameters A = 'sbuf+len' and *
                >425  *  X = 'sbuf+len+1'.                                    *
                >426  *                                                       *
                >427  *  CALLREQ, like other requests, will retry the request  *
                >428  *  in case of error, up to 'maxreqrt' times.  If errors  *
                >429  *  persist, it will return with Carry set.              *
                >430  *                                                       *
                >431  *  CALLREQ does the following steps:                    *
                >432  *     1. Make the CALL request (and receive the ACK)    *
                >433  *     2. Retry in case of error up to 'maxreqrt' times  *
                >434  *                                                       *
                >435  *********************************************************
                >436
961B: A9 03     >437  CALLREQ  lda   #maxreqrt  ; Set request retry
961D: 8D 51 91  >438           sta   reqretry   ;  counter.
9620: A9 18     >439  :retry   lda   #r_CALL    ; Send CALL request.
9622: 20 3A 96  >440           jsr   REQUEST
9625: 90 06     >441           bcc   :done
9627: CE 51 91  >442  :failed  dec   reqretry   ; Dec request retry count
962A: D0 F4     >443           bne   :retry     ; Try until OK or exhausted,
962C: 38        >444           sec              ;  then return with C set.
962D: 60        >445  :done    rts
```

```
                >449  ************************************************************
                >450  *                                                          *
                >451  *                      C A L L S R V                       *
                >452  *                                                          *
                >453  *             Michael J. Mahon - May 11, 1996              *
                >454  *                   Revised Sep 25, 2008                   *
                >455  *                                                          *
                >456  *              Copyright (c) 1996, 2004, 2008              *
                >457  *                                                          *
                >458  *  Service machine 'rbuf+frm's request to call a           *
                >459  *  subroutine at our 'rbuf+adr' with parameters             *
                >460  *  A = 'rbuf+len' and X = 'rbuf+len+1'.  Flags are set      *
                >461  *  according to the value of A.                             *
                >462  *                                                          *
                >463  *  Note that when the subroutine returns, it returns to    *
                >464  *  whoever called SERVER.                                   *
                >465  *                                                          *
                >466  *  CALLSRV does the following steps:                        *
                >467  *     1. Send the ACK packet                               *
                >468  *     2. Load parameters from 'rbuf+len' into A and X       *
                >469  *     3. Call subroutine at 'rbuf+adr'                     *
                >470  *                                                          *
                >471  ************************************************************
                >472
962E: 20 12 98  >473  CALLSRV  jsr   SENDACK    ; ACK the request.
9631: AE 49 91  >474  docall   ldx   rbuf+len+1 ; Set X parameter
9634: AD 48 91  >475           lda   rbuf+len   ;  and A parameter, and
9637: 6C 46 91  >476           jmp   (rbuf+adr) ; Jump to requested address.
```

```
                        >480  ************************************************************
                        >481  *                                                          *
                        >482  *                      R E Q U E S T                       *
                        >483  *                                                          *
                        >484  *             Michael J. Mahon - April 20, 2004            *
                        >485  *                   Revised Aug 17, 2008                   *
                        >486  *                                                          *
                        >487  *              Copyright (c) 1996, 2004, 2008              *
                        >488  *                                                          *
                        >489  *  Handle request protocol for the request in A & 'sbuf'.*
                        >490  *                                                          *
                        >491  *  Retry the protocol for up to 'reqtime' ms. (up to       *
                        >492  *  'retrylim' times).  If successful, return with valid    *
                        >493  *  response in 'rbuf' and Carry clear.                     *
                        >494  *                                                          *
                        >495  *  If request timed out, return with Carry set and A=0.    *
                        >496  *                                                          *
                        >497  *  If NAK received, return with Carry set and A>0.         *
                        >498  *                                                          *
                        >499  *  REQUEST performs the following steps:                   *
                        >500  *     1. Complete control pkt in 'sbuf' (request in A)     *
                        >501  *     2. Arbitrate for the use of the bus                  *
                        >502  *     3. Send the request specified in 'sbuf'              *
                        >503  *     4. Receive the control response into 'rbuf'          *
                        >504  *     5. Check 'rbuf' for a valid, expected response       *
                        >505  *     6. Retry steps 2 to 5 up to 'retrylim' times         *
                        >506  *     7. When ACKed, NAKed, or timed-out, return           *
                        >507  *                                                          *
                        >508  ************************************************************
                        >509
963A: 09 01             >510  REQUEST  ora   #rm_REQ    ; Add REQ modifier and
963C: 8D 3A 91          >511           sta   sbuf+rqmd  ; Store request code.
963F: AD 4C 91          >512           lda   retrylim   ; Init retry counter.
9642: 8D 52 91          >513           sta   retrycnt
9645: AD 52 91          >514  :retry   lda   retrycnt   ; Timed out?
9648: F0 4A             >515           beq   :err       ; -Yes, return w/ C set, A=0
964A: CE 52 91          >516           dec   retrycnt   ; Dec retry counter.
964D: 20 00 98          >517           jsr   ARBTRATE   ; Arbitrate for & lock bus
9650: 20 26 98          >518           jsr   SENDCTL    ; Send request in 'sbuf'.
9653: 20 00 99          >519           jsr   RCVCTL     ; Receive response in 'rbuf'.
9656: 90 0D             >520           bcc   :ok        ; -Clean packet received.
                        >521           dlyms reqdelay   ;  delay a few ms.
9658: A0 11             >521           ldy   #reqdelay  ; Delay 1ms. per iteration
                        >521  ]dly     delay 1020-4     ; Cycles per ms. - 4
965A: A2 CB             >521           ldx   #1020-4/5  ; (5 cycles per iteration)
965C: CA                >521  ]delay   dex
965D: D0 FD             >521           bne   ]delay
                        >521           eom
965F: 88                >521           dey
9660: D0 F8             >521           bne   ]dly
                        >521           eom
9662: 4C 45 96          >522           jmp   :retry     ;    and try again...
                        >523
```

```
9665: AD 44 91 >524  :ok      lda   rbuf+dst   ; Message received, is
9668: CD 39 91 >525           cmp   self       ;  it for us?
966B: D0 1D    >526           bne   :proterr   ; -No, error.
966D: AD 3C 91 >527           lda   sbuf+dst   ; -Yes.  Is it from
9670: CD 45 91 >528           cmp   rbuf+frm   ;   our destination?
9673: D0 15    >529           bne   :proterr   ; -No.  Protocol error.
9675: AD 3A 91 >530           lda   sbuf+rqmd  ; -Yes.  Is the
9678: 29 F8    >531           and   #reqmask   ;   modifier field
967A: 09 02    >532           ora   #rm_ACK    ;    'ACK'?
967C: CD 42 91 >533           cmp   rbuf+rqmd  ;      as expected?
967F: F0 0F    >534           beq   :good      ; -Yes, good response!
9681: 29 F8    >535           and   #reqmask   ; -No, construct
9683: 09 04    >536           ora   #rm_NAK    ;   the 'NAK' value.
9685: CD 42 91 >537           cmp   rbuf+rqmd  ; Is it a NAK?
9688: F0 08    >538           beq   :nakexit   ; -Yes, return w/ C set, A=1
968A: 20 8F 99 >539  :proterr jsr   PROTERR    ; -No, count protocol errors.
968D: 4C 45 96 >540           jmp   :retry     ;    and try again...
               >541
9690: 18       >542  :good    clc              ; Signal good ACK
9691: 60       >543           rts              ;  and return.
               >544
9692: A9 01    >545  :nakexit lda   #1         ; Signal NAK
9694: 38       >546  :err     sec              ; Signal error
9695: 60       >547           rts              ;  and return.
```

```
                        >551  ************************************************************
                        >552  *                                                          *
                        >553  *                      R C V D A C K                       *
                        >554  *                                                          *
                        >555  *             Michael J. Mahon - Apr 19, 2004              *
                        >556  *                   Revised Aug 17, 2008                   *
                        >557  *                                                          *
                        >558  *                Copyright (c) 2004, 2008                  *
                        >559  *                                                          *
                        >560  *  Receive DATA ACK packet.  Require a good cksum,         *
                        >561  *  addressed to us, response req = sent req.  If all OK,   *
                        >562  *  return with Carry clear, else with Carry set.           *
                        >563  *                                                          *
                        >564  ************************************************************
                        >565
9696: 20 00 99          >566  RCVDACK  jsr   RCVCTL     ; Receive response packet.
9699: B0 1E             >567           bcs   :err        ; Cksum error or timeout.
969B: AD 44 91          >568  :ok      lda   rbuf+dst    ; Is packet for us?
969E: CD 39 91          >569           cmp   self
96A1: D0 18             >570           bne   :proterr    ; -No, protocol error.
96A3: AD 45 91          >571           lda   rbuf+frm    ; Was it sent by receiver?
96A6: CD 3C 91          >572           cmp   sbuf+dst
96A9: D0 10             >573           bne   :proterr    ; -No, protocol error.
96AB: AD 3A 91          >574           lda   sbuf+rqmd   ; Construct sent req
96AE: 29 F8             >575           and   #reqmask    ;  with the expected
96B0: 09 03             >576           ora   #rm_DACK    ;   'DACK' modifier.
96B2: CD 42 91          >577           cmp   rbuf+rqmd   ; Does it match?
96B5: D0 04             >578           bne   :proterr    ; -No, protocol error.
96B7: 18                >579           clc               ; -Yes, clear Carry
96B8: 60                >580  :return  rts               ;   and return.
                        >581
96B9: D0 FD             >582  :err     bne   :return     ; Cksum error return.
96BB: 38                >583  :proterr sec               ; Return with C set,
96BC: 4C 8F 99          >584           jmp   PROTERR     ;  after counting error.
```

```
 59           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
96BF: A9 03     >79   PUTMREQ  lda   #maxreqrt  ; Set request retry
96C1: 8D 51 91  >80            sta   reqretry   ;  counter.
96C4: A9 20     >81   :retry   lda   #r_PUTMSG  ; Send PUTMSG request.
96C6: 20 3A 96  >82            jsr   REQUEST
96C9: B0 0D     >83            bcs   :failed
96CB: 20 E9 98  >84            jsr   lasl=>al   ; Set up address/length
96CE: 20 B0 99  >85            jsr   SENDLONG   ;  and send message.
96D1: 20 96 96  >86            jsr   RCVDACK    ; Receive DATA ACK packet.
96D4: 90 0A     >87            bcc   :done      ; -All OK.
96D6: A9 00     >88            lda   #0         ; Not a NAK error
96D8: D0 05     >89   :failed  bne   :nakexit
96DA: CE 51 91  >90   :cksumer dec   reqretry   ; Dec request retry count
96DD: D0 E5     >91            bne   :retry     ; Try until OK or exhausted,
96DF: 38        >92   :nakexit sec               ;  then return with C set.
96E0: 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
96E1: A9 03             >125  GETMREQ  lda   #maxreqrt  ; Set request retry
96E3: 8D 51 91          >126          sta   reqretry   ;  counter.
96E6: A9 28             >127  :retry   lda   #r_GETMSG  ; Send GETMSG request.
96E8: 20 3A 96          >128          jsr   REQUEST
96EB: B0 1C             >129          bcs   :failed    ; Timeout or no msg.
96ED: 20 F3 98          >130          jsr   la=>a      ; Set up address
                        >131          mov16 rbuf+len;length ; and length.
96F0: AD 48 91          >131          lda   rbuf+len   ; Move 2 bytes
96F3: 85 FE             >131          sta   length
96F5: AD 49 91          >131          lda   1+rbuf+len
96F8: 85 FF             >131          sta   1+length
                        >131          eom
96FA: 20 CF 99          >132          jsr   RCVLONG    ; Receive segmented message
96FD: B0 0C             >133          bcs   :err       ; Timeout or cksum err.
                        >134          delay 40         ; Kill some time...
96FF: A2 08             >134          ldx   #40/5      ; (5 cycles per iteration)
9701: CA                >134  ]delay   dex
9702: D0 FD             >134          bne   ]delay
                        >134          eom
9704: A9 03             >135          lda   #rm_DACK   ; -OK, send DATA ACK.
9706: 4C 14 98          >136          jmp   SENDRSP    ;   and return w/ C clear.
                        >137
9709: D0 05             >138  :failed  bne   :nak       ; Server has no message.
```

```
970B: CE 51 91 >139  :err     dec   reqretry   ; Cksum or timeout; dec count.
970E: D0 D6    >140           bne   :retry     ; Try until OK or exhausted,
9710: 38       >141  :nak     sec              ;  then return with C set.
9711: 60       >142           rts
```

```
 60           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
9712: 20 00 98  >68   BCASTARB jsr   ARBTRATE   ; Arbitrate and lock network.
                >69           dlyms 20         ; Let collisions resolve.
9715: A0 14     >69           ldy   #20        ; Delay 1ms. per iteration
                >69   ]dly    delay 1020-4     ; Cycles per ms. - 4
9717: A2 CB     >69           ldx   #1020-4/5  ; (5 cycles per iteration)
9719: CA        >69   ]delay  dex
971A: D0 FD     >69           bne   ]delay
                >69           eom
971C: 88        >69           dey
971D: D0 F8     >69           bne   ]dly
                >69           eom
971F: A9 00     >70           lda   #0         ; Set broadcast
9721: 8D 3C 91  >71           sta   sbuf+dst   ;  request.
9724: 60        >72           rts              ;  and return.
```

```
                >76   ]end     align 256        ; Align to next page.
9725: 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
9800: AE 4E 91  >94   ARBTRATE ldx   arbxv      ; Set arbitration wait.
9803: CD E8 C0  >95             cmp   zipslow    ; Zip Chip to 1MHz mode.
9806: 2C 62 C0  >96   :waitidl  bit   drecv      ; Wait for idle bus.
9809: 30 F5     >97             bmi   ARBTRATE   ; Restart timing.
980B: CA        >98             dex
980C: D0 F8     >99             bne   :waitidl   ; ...not yet.
980E: 8D 5B C0 >100             sta   dsend+1    ; Got it!  Lock the bus
9811: 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
9812: A9 02             >145  SENDACK  lda   #rm_ACK    ; Build an ACK packet
9814: 85 EC             >146  SENDRSP  sta   ckbyte     ; Store modifier for ora
9816: AD 42 91          >147           lda   rbuf+rqmd  ; Get received request
9819: 29 F8             >148           and   #reqmask   ;  isolate request
981B: 05 EC             >149           ora   ckbyte     ;   and OR in modifier.
981D: 8D 3A 91          >150           sta   sbuf+rqmd  ; Set response code.
9820: AD 45 91          >151           lda   rbuf+frm
9823: 8D 3C 91          >152           sta   sbuf+dst   ; Destination (= requester)
9826: A9 3A             >153  SENDCTL  lda   #<sbuf     ; Control pkt send buffer
9828: A0 91             >154           ldy   #>sbuf
982A: A2 08             >155           ldx   #lenctl
```

```
                      >156
982C: CD E8 C0        >158  SENDPKT  cmp   zipslow    ; Slow Zip Chip for packet.
982F: 8D 5B C0        >162           sta   dsend+1    ; Send start signal ONE
9832: 20 9B 98        >163           jsr   :exit      ; Stretch it.
9835: 86 EC           >164           stx   ckbyte     ; Seed ckbyte with length.
9837: 84 EE           >165           sty   ptr+1      ; Y = start address hi
9839: A0 00           >166           ldy   #0         ; Index first data byte
983B: 18              >167           clc              ; Ensure C clear at exit
983C: 08              >168           php              ; Save interrupt state
983D: 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
983E: 8D 5A C0        >175           sta   dsend+0    ; Send start sync ZERO
9841: 2C 9B 98        >176           bit   :exit      ; Set V to send ckbyte at end
9844: 85 ED           >177           sta   ptr        ; A = start address lo
9846: B1 ED           >178           lda   (ptr),y    ; Get first data (Y=0 so no px)
9848: 8D 5B C0        >179           sta   dsend+1    ; Send coarse sync at t=16
984B: EA              >180           nop
984C: 38              >181           sec              ; Ensure C set between bytes
984D: 99 5A C0        >182           sta   dsend+0,y  ; Release coarse sync at t=24
9850: B0 03           >183           bcs   :servo     ; Go send servo at t=32
                      >184
9852: C8              >185  :t84v0   iny              ; Get next data byte and
9853: B1 ED           >186           lda   (ptr),y    ;  send servo at t=94 or 95
                      >187
9855: 8D 5B C0        >188  :servo   sta   dsend+1    ; Servo ONE
9858: 2A              >189           rol
9859: 90 41           >190           bcc   :t06b7v1
985B: 8D 5A C0        >191           sta   dsend+0    ; Bit 7 ZERO at t=8
985E: 2A              >192           rol
985F: B0 3D           >193           bcs   :t14b6v0
9861: 8D 5B C0        >194           sta   dsend+1    ; Bit 6 ONE at t=16
9864: 2A              >195  :t17b6v1 rol
9865: 90 39           >196           bcc   :t22b5v1
9867: 8D 5A C0        >197           sta   dsend+0    ; Bit 5 ZERO at t=24
986A: 2A              >198  :t25b5v0 rol
986B: B0 35           >199           bcs   :t30b4v0
986D: 8D 5B C0        >200           sta   dsend+1    ; Bit 4 ONE at t=32
9870: 2A              >201  :t33b4v1 rol
9871: 90 31           >202           bcc   :t38b3v1
9873: 8D 5A C0        >203           sta   dsend+0    ; Bit 3 ZERO at t=40
9876: 2A              >204  :t41b3v0 rol
9877: B0 2D           >205           bcs   :t46b2v0
9879: 8D 5B C0        >206           sta   dsend+1    ; Bit 2 ONE at t=48
987C: 2A              >207  :t49b2v1 rol
987D: 90 29           >208           bcc   :t54b1v1
987F: 8D 5A C0        >209           sta   dsend+0    ; Bit 1 ZERO at t=56
9882: 2A              >210  :t57b1v0 rol
9883: B0 25           >211           bcs   :t62b0v0
9885: 8D 5B C0        >212           sta   dsend+1    ; Bit 0 ONE at t=64
```

```
9888: 2A       >213  :t65b0v1 rol             ; Restore data, set C
9889: EA       >214           nop
988A: 8D 5A C0 >215           sta   dsend+0   ; Idle/interbyte ZERO at t=72
               >216
988D: 45 EC    >217  :t73v0   eor   ckbyte    ; Compute checksum
988F: 85 EC    >218           sta   ckbyte    ;  and save it.
9891: CA       >219           dex             ; Count bytes sent
9892: D0 BE    >220           bne   :t84v0    ; Loop while more to send
               >221
9894: 50 04    >222  :t83v0   bvc   :done     ; Quit if ckbyte already sent;
9896: E8       >223           inx             ;  else count ckbyte,
9897: B8       >224           clv             ;  clear send-ckbyte flag,
9898: 50 BB    >225           bvc   :servo    ; Send ckbyte servo at t=95
               >226
989A: 28       >227  :done    plp             ; Restore int state
989B: 60       >228  :exit    rts             ;  and return with C clear.
               >229
989C: 90 1E    >230  :t06b7v1 bcc   :t09b7v1  ; These are all for timing
989E: B0 22    >231  :t14b6v0 bcs   :t17b6v0  ;  equalization and all of
98A0: 90 26    >232  :t22b5v1 bcc   :t25b5v1  ;  them are always taken
98A2: B0 2A    >233  :t30b4v0 bcs   :t33b4v0
98A4: 90 2E    >234  :t38b3v1 bcc   :t41b3v1
98A6: B0 32    >235  :t46b2v0 bcs   :t49b2v0
98A8: 90 36    >236  :t54b1v1 bcc   :t57b1v1
98AA: B0 3A    >237  :t62b0v0 bcs   :t65b0v0
               >238
98AC: 90 B6    >239  :t14b6v1 bcc   :t17b6v1
98AE: B0 BA    >240  :t22b5v0 bcs   :t25b5v0
98B0: 90 BE    >241  :t30b4v1 bcc   :t33b4v1
98B2: B0 C2    >242  :t38b3v0 bcs   :t41b3v0
98B4: 90 C6    >243  :t46b2v1 bcc   :t49b2v1
98B6: B0 CA    >244  :t54b1v0 bcs   :t57b1v0
98B8: 90 CE    >245  :t62b0v1 bcc   :t65b0v1
98BA: B0 D1    >246  :t70v0   bcs   :t73v0
               >247
98BC: 2A       >248  :t09b7v1 rol
98BD: 90 ED    >249           bcc   :t14b6v1
98BF: 8D 5A C0 >250           sta   dsend+0   ; Bit 6 ZERO at t=16
98C2: 2A       >251  :t17b6v0 rol
98C3: B0 E9    >252           bcs   :t22b5v0
98C5: 8D 5B C0 >253           sta   dsend+1   ; Bit 5 ONE at t=24
98C8: 2A       >254  :t25b5v1 rol
98C9: 90 E5    >255           bcc   :t30b4v1
98CB: 8D 5A C0 >256           sta   dsend+0   ; Bit 4 ZERO at t=32
98CE: 2A       >257  :t33b4v0 rol
98CF: B0 E1    >258           bcs   :t38b3v0
98D1: 8D 5B C0 >259           sta   dsend+1   ; Bit 3 ONE at t=40
98D4: 2A       >260  :t41b3v1 rol
98D5: 90 DD    >261           bcc   :t46b2v1
98D7: 8D 5A C0 >262           sta   dsend+0   ; Bit 2 ZERO at t=48
98DA: 2A       >263  :t49b2v0 rol
98DB: B0 D9    >264           bcs   :t54b1v0
98DD: 8D 5B C0 >265           sta   dsend+1   ; Bit 1 ONE at t=56
```

```
98E0: 2A       >266  :t57b1v1 rol
98E1: 90 D5    >267           bcc   :t62b0v1
98E3: 8D 5A C0 >268           sta   dsend+0    ; Bit 0 ZERO at t=64
98E6: 2A       >269  :t65b0v0 rol              ; Restore data, set C
98E7: 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
98E9: AD 40 91  >280           lda   sbuf+len   ; Move 2 bytes
98EC: 85 FE     >280           sta   length
98EE: AD 41 91  >280           lda   1+sbuf+len
98F1: 85 FF     >280           sta   1+length
                >280           eom
                >281  la=>a    mov16 locaddr;address ; Local address -> address
98F3: AD 4A 91  >281           lda   locaddr    ; Move 2 bytes
98F6: 85 FC     >281           sta   address
98F8: AD 4B 91  >281           lda   1+locaddr
98FB: 85 FD     >281           sta   1+address
                >281           eom
98FD: 60        >282           rts
```

```
                >284  ]end     align 256        ; Align to next page.
98FE: 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
9900: A9 42     >405  RCVCTL   lda   #<rbuf     ; Receive control pkt to 'rbuf'
9902: A0 91     >406           ldy   #>rbuf
9904: A2 08     >407           ldx   #lenctl
9906: 85 ED     >408  RCVPKT   sta   ptr        ; A = buf address lo
9908: 84 EE     >409           sty   ptr+1      ; Y = buf address hi
990A: 8A        >410  RCVPTR   txa              ; Seed checksum with length
990B: CA        >411           dex              ; X = length 1..256 (0=>256);
990C: 86 EB     >412           stx   lastidx    ;  convert to last buffer index
                >413
990E: AC 4F 91  >414           ldy   tolim      ; Wait <= (tolim-1) * 2.8ms.
9911: A2 5C     >415           ldx   #arbx      ;  plus minimum arb time.
9913: 08        >416           php              ; Save interrupt state
9914: 78        >417           sei              ;  and disable interrupts.
9915: 2C E8 C0  >418           bit   zipslow    ; Slow any Zip Chip to 1 MHz.
                >419
9918: 2C 62 C0  >420  :waitstr bit   drecv      ; Wait for starting ONE.
991B: 30 0A     >421           bmi   :gotstr
991D: CA        >422           dex              ; (inner loop is 11 cycles)
991E: D0 F8     >423           bne   :waitstr   ; Keep waiting...
9920: 88        >424           dey              ; (outer loop is 2820 cycles)
9921: D0 F5     >425           bne   :waitstr   ; Loop for 'timeout' ms.
                >426
9923: 28        >427           plp              ; Restore int state
9924: 98        >428           tya              ; Signal timeout (A=0, Z set)
9925: 38        >429           sec              ;  and return with C set.
9926: 60        >430  :exit    rts
                >431
9927: 2C E8 C0  >432  :gotstr  bit   zipslow    ; Slow Zip Chip for packet.
                >433
992A: 2C 62 C0  >434  :waitsyn bit   drecv      ; Wait for 16-cycle sync ZERO;
992D: 30 FB     >435           bmi   :waitsyn   ;  too bad if bus locks forever!
                >436
992F: A0 FF     >437           ldy   #$FF       ; Index-1 of first data location
9931: A2 7F     >438           ldx   #$7F       ; CPX #0-7F sets C, 80-FF clears
9933: EC 62 C0  >439  :synrt07 cpx   drecv      ; Check for coarse sync at rt=10
9936: B0 05     >440           bcs   :synrt14   ; Only do delay if still ZERO
                >441
```

```
9938: 2C 26 99 >442  :synrt19 bit   :exit      ; Set V (not-ckbyte flag)
993B: 70 04    >443           bvs   :servo     ; Do first servo check at rt=29
               >444
993D: 18       >445  :synrt14 clc             ; 6-cycle coarse sync delay
993E: 90 F8    >446           bcc   :synrt19   ; (1 extra to get here, 5 back)
               >447
9940: B8       >448  :rt88    clv             ; Clear not-ckbyte flag
               >449
9941: EC 62 C0 >450  :servo   cpx   drecv      ; Check for servo transition
9944: 90 02    >451  :rt01    bcc   :rt04      ; Delay 3 cyc if past servo,
9946: EA       >452           nop             ;  6 if not
9947: EA       >453           nop
               >454
9948: 85 EC    >455  :rt04    sta   ckbyte     ; Update checksum
994A: C8       >456           iny             ; Index next data location
               >457
994B: EC 62 C0 >458  :rt09    cpx   drecv      ; C <-- ~ bit 7 at rt=12
994E: 2A       >459           rol             ; Shift bit 7 in.
994F: EA       >460           nop
9950: EC 62 C0 >461           cpx   drecv      ; C <-- ~ bit 6 at rt=20
9953: 2A       >462           rol
9954: EA       >463           nop
9955: EC 62 C0 >464           cpx   drecv      ; C <-- ~ bit 5 at rt=28
9958: 2A       >465           rol
9959: EA       >466           nop
995A: EC 62 C0 >467           cpx   drecv      ; C <-- ~ bit 4 at rt=36
995D: 2A       >468           rol
995E: EA       >469           nop
995F: EC 62 C0 >470           cpx   drecv      ; C <-- ~ bit 3 at rt=44
9962: 2A       >471           rol
9963: EA       >472           nop
9964: EC 62 C0 >473           cpx   drecv      ; C <-- ~ bit 2 at rt=52
9967: 2A       >474           rol
9968: EA       >475           nop
9969: EC 62 C0 >476           cpx   drecv      ; C <-- ~ bit 1 at rt=60
996C: 2A       >477           rol
996D: EA       >478           nop
996E: EC 62 C0 >479           cpx   drecv      ; C <-- ~ bit 0 at rt=68
9971: 2A       >480  :rt69    rol
9972: 50 0A    >481  :rt71    bvc   :rcvdone   ; quit after ckbyte
               >482
9974: 91 ED    >483  :rt73    sta   (ptr),y    ; Save data (always 6cy)
9976: 45 EC    >484  :rt79    eor   ckbyte     ; Compute checksum
9978: C4 EB    >485  :rt82    cpy   lastidx    ; Stored last byte?
997A: F0 C4    >486  :rt85    beq   :rt88      ; Go clear not-ckbyte flag if so
997C: D0 C3    >487  :rt87    bne   :servo     ; Do next servo sample at rt=93
               >488
997E: 45 EC    >489  :rcvdone eor   ckbyte     ; A = 0 if ckbyte = sum
9980: F0 08    >490           beq   :goodck    ; -No error.
               >491           inc16 ckerr      ; Count checksum error.
9982: EE 55 91 >491           inc   ckerr      ; Increment 16-bit word.
9985: D0 03    >491           bne   *+5        ; - No carry.
9987: EE 56 91 >491           inc   ckerr+1    ; Propagate carry.
```

```
                >491            eom
998A: 28        >492  :goodck   plp             ; Restore int state
998B: C9 01     >493            cmp   #1        ; Set C & NZ if checksum bad,
998D: AA        >494            tax             ;  clear C and set Z if good
998E: 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.
998F: EE 53 91  >503           inc   errprot    ; Increment 16-bit word.
9992: D0 03     >503           bne   *+5        ; - No carry.
9994: EE 54 91  >503           inc   errprot+1  ; Propagate carry.
                >503           eom
9997: 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
9998: AD 48 91  >515           lda   rbuf+len   ; Move 2 bytes
999B: 85 FE     >515           sta   length
999D: AD 49 91  >515           lda   1+rbuf+len
99A0: 85 FF     >515           sta   1+length
                >515           eom
                >516  ra=>a    mov16 rbuf+adr;address; 'rbuf' address -> address
99A2: AD 46 91  >516           lda   rbuf+adr   ; Move 2 bytes
99A5: 85 FC     >516           sta   address
99A7: AD 47 91  >516           lda   1+rbuf+adr
99AA: 85 FD     >516           sta   1+address
                >516           eom
99AC: 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
99AD: CA        >543  DSENDLNG dex             ; Delay 5 * X - 1 cycles
99AE: D0 FD     >544           bne   DSENDLNG  ;  and fall into SENDLONG.
99B0: A5 FF     >545  SENDLONG lda   length+1  ; How many 256-byte pages?
99B2: F0 0F     >546           beq   :short    ; - None, just a short pkt.
99B4: A2 00     >547  :loop    ldx   #0        ; Set 256 byte packet.
99B6: A5 FC     >548           lda   address   ;  and point to
99B8: A4 FD     >549           ldy   address+1 ;   data buffer.
99BA: 20 2C 98  >550           jsr   SENDPKT   ; Send 256 bytes.
99BD: E6 FD     >551           inc   address+1 ; Advance to next page
99BF: C6 FF     >552           dec   length+1  ;  and decrement page
99C1: D0 F1     >553           bne   :loop     ;   count until done...
99C3: A6 FE     >554  :short   ldx   length    ; Remaining data length.
99C5: F0 07     >555           beq   :done     ; -All done.
99C7: A5 FC     >556           lda   address
99C9: A4 FD     >557           ldy   address+1
99CB: 20 2C 98  >558           jsr   SENDPKT   ; Send the final packet.
99CE: 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
99CF: A5 FF     >585  RCVLONG  lda   length+1   ; How many 256-byte pages?
99D1: F0 11     >586           beq   :short     ; - None, just a short pkt.
99D3: A2 00     >587  :loop    ldx   #0          ; Set 256 byte packet.
99D5: A5 FC     >588           lda   address     ;  and point to
99D7: A4 FD     >589           ldy   address+1   ;   data buffer.
99D9: 20 06 99  >590           jsr   RCVPKT      ; Receive 256 bytes.
99DC: B0 14     >591           bcs   :err        ; Receive error detected.
99DE: E6 FD     >592           inc   address+1   ; Advance to next page
99E0: C6 FF     >593           dec   length+1    ;  and decrement page
99E2: D0 EF     >594           bne   :loop       ;   count until done...
99E4: A6 FE     >595  :short   ldx   length      ; Remaining data length.
99E6: F0 09     >596           beq   :done       ; -All done.
99E8: A5 FC     >597           lda   address
99EA: A4 FD     >598           ldy   address+1
99EC: 20 06 99  >599           jsr   RCVPKT      ; Receive final packet.
99EF: B0 01     >600           bcs   :err        ; Keep track of any errors.
99F1: 60        >601  :done    rts
                >602
99F2: D0 03     >603  :err     bne   :ckerr      ; Checksum error.
99F4: 20 8F 99  >604           jsr   PROTERR     ; Count protocol error.
99F7: A8        >605  :ckerr   tay               ; Set Z flag from A.
99F8: 60        >606           rts
```

```
                61   ]end     align 256        ; Align to page boundary
99F9: 00 00 00  61            ds    *-1/256*256+256-*
                61            eom
                62   xreceive equ   *-]end     ; Extra space at end.
                63            err   *-1-entry/SIZE ; Can't exceed limit


--End assembly, 2304 bytes, Errors: 0


Symbol table - alphabetical order:

   @       =$9138      ADDON   =$D998      AMPNADA =$9247      AMPVECT =$03F5
   ARBTRATE=$9800      AX      =$48        BCASTARB=$9712      BCASTREQ=$9497
?  BELL    =$FF3A      BOOTREQ =$948E      BPOKEREQ=$9601      BPOKESRV=$960C
   BRUNREQ =$9565      BRUNSRV =$95A6      CALLREQ =$961B      CALLSRV =$962E
   CALL_t  =$8C        CHRGET  =$B1        CHRGOT  =$B7        COLDSTRT=$E000
   COUT    =$FDED      CROUT1  =$FD8B      CSW     =$36        DSENDLNG=$99AD
   ERROR   =$D412      FAC     =$9D        FIXLINKS=$D4F2      FLO2    =$EBA0
   FORPNT  =$85        FRETOP  =$6F        FRMNUM  =$DD67      GETADR  =$E752
   GETBYT  =$E6F8      GETIDSRV=$94B2      GETMREQ =$96E1      GET_t   =$BE
   HIMEM   =$73     ?  HOME    =$FC58      INIT    =$93A8      INSTALL =$921C
   INTFLG  =$12        KSW     =$38        ONERR   =$D8        PEEKREQ =$94E1
   PEEKSRV =$9512      PEEK_t  =$E2        PKINCREQ=$9535      PKINCSRV=$9548
   POKEREQ =$9569      POKESRV =$95A6      POKE_t  =$B9     ?  PRBL2   =$F94A
   PRBYTE  =$FDDA   ?  PREAD   =$FB1E   ?  PROGEND =$AF        PROTERR =$998F
   PSTART  =$67        PTRGET  =$DFE3      PUTMREQ =$96BF   ?  PWREDUP =$03F4
?  RARL=>AL=$9133      RCVCTL  =$9900      RCVDACK =$9696      RCVLONG =$99CF
   RCVPKT  =$9906      RCVPTR  =$990A      REQUEST =$963A      ROMboot =$00
   RUNPROG =$D566      RUNREQ  =$9561      RUNSRV  =$9591      RUN_t   =$AC
   SENDACK =$9812      SENDCTL =$9826      SENDLONG=$99B0      SENDPKT =$982C
   SENDRSP =$9814      SERVER  =$93F7      SETFOR  =$EB27      SIZE    =$0900
?  SOFTEV  =$03F2      SYNCHR  =$DEC0      SYNERR  =$DEC9      TXTPTR  =$B8
   VALTYP  =$11        VARTAB  =$69     ?  VBL     =$C019   V? ]PROTERR=$9462
V  ]cpx    =$0B     V  ]cpy    =$0B04   V  ]cy     =$4FB0   MV ]delay  =$9719
MV ]dly    =$9717   V? ]doit   =$9499   V  ]end    =$99F9   V  ]servpad=$FF
   addr    =$46        address =$FC        adr     =$04     MD align   =$8000
   an0     =$C058      an1     =$C05A   ?  an2     =$C05C   ?  an3     =$C05E
   arbtime =$01        arbx    =$5C        arbxv   =$914E   ?  bcast   =$911E
   bootself=$03CC   ?  bpoke   =$9121   ?  brun    =$912A      byte    =$00
?  call    =$9115      chain   =$925D      ckbyte  =$EC        ckerr   =$9155
   class   =$46        cmd     =$9254      cmdptr  =$EC        cmdsave =$ED
   cmdtable=$9181      comp    =$9260      crate   =$00        cyperms =$03FC
MD delay   =$8000      dest    =$04        disp    =$EF     MD dlyms   =$8000
   docall  =$9631      dos     =$00        drecv   =$C062      dsend   =$C05A
   dsk6off =$C0E8      dst     =$02        enhboot =$00        entry   =$9100
   errprot =$9153      errstop =$917C      frm     =$03        frmc    =$01
   frmcerr =$9157   ?  gapwait =$07     ?  getmsg  =$911B      idletime=$14
   idleto  =$08        idtable =$915A      idtbl   =$9371   MD inc16   =$8000
   incr    =$48     ?  init    =$9109      instald =$917A      iter    =$15
   kbstrobe=$C010      keybd   =$C000      la=>a   =$98F3      lasl=>al=$98E9
   lastidx =$EB        len     =$06        lenctl  =$08        length  =$FE
   lngth   =$48        lngth?  =$D0        locaddr =$914A      locadr  =$52
```

```
   master  =$01     ?  maxarb  =$03        maxgap  =$57        maxid   =$1F
   maxreq  =$68        maxreqrt=$03        maxretry=$32        modmask =$07
MD mov16   =$8000      mserve  =$00        n60ms   =$14        nadapage=$03CF
?  nadaver =$9159      nparms  =$917B      null    =$936E      parmsiz =$15
   pb0     =$C061      pb1     =$C062   ?  pb2     =$C063   ?  peek    =$910F
?  peekinc =$9124   ?  poke    =$9112      ptr     =$ED     ?  ptrig   =$C070
?  putmsg  =$9118      r_BCAST =$40        r_BOOT  =$38        r_BPOKE =$48
   r_BRUN  =$60        r_CALL  =$18     ?  r_GETID =$30        r_GETMSG=$28
   r_PEEK  =$08        r_PKINC =$50        r_POKE  =$10        r_PUTMSG=$20
   r_RUN   =$58        ra=>a   =$99A2      rarl=>al=$9998      rbuf    =$9142
?  rcvctl  =$912D   ?  rcvlong =$9136   ?  rcvptr  =$9130      reqctr  =$9150
   reqdelay=$11        reqdur  =$06        reqfac  =$08        reqmask =$F8
   reqpidle=$03        reqretry=$9151      reqtime =$0BB8      reqto   =$01
   retrycnt=$9152      retrylim=$914C      rm_ACK  =$02        rm_DACK =$03
   rm_NAK  =$04        rm_REQ  =$01        rqmd    =$00        rqperiod=$14
   rts     =$9600   ?  run     =$9127      savhooks=$02FC      sbuf    =$913A
   self    =$9139   ?  serve   =$910C      servecnt=$914D      servegap=$3A
   servelp =$9103      service =$937D      sethooks=$958D      setid   =$93DA
   setreq  =$956B   ?  spkr    =$C030      svrxkbd =$93F4   ?  t_BASIC =$E0
?  t_SYNTH =$F0     ?  t_VOICE =$F1        timeout =$9364      tolim   =$914F
   val     =$48        val?    =$D0        var     =$80        varadr  =$917F
   varcmd  =$917D      vartype =$917E      verlen  =$13        vermsg  =$9395
   version =$30        warmstrt=$03CD      word    =$40     ?  xmain   =$DB
?  xreceive=$07     ?  xsend   =$02        zipslow =$C0E8
```

Symbol table - numerical order:

```
   dos     =$00        crate   =$00        mserve  =$00        ROMboot =$00
   enhboot =$00        rqmd    =$00        byte    =$00        master  =$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
   maxid   =$1F        r_PUTMSG=$20        r_GETMSG=$28        version =$30
?  r_GETID =$30        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_RUN   =$58        arbx    =$5C
   r_BRUN  =$60        PSTART  =$67        maxreq  =$68        VARTAB  =$69
   FRETOP  =$6F        HIMEM   =$73        var     =$80        FORPNT  =$85
   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   =$D8
?  xmain   =$DB     ?  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    =$0900
V  ]cpy    =$0B04     reqtime =$0BB8   V  ]cy     =$4FB0      entry   =$9100
   servelp =$9103   ?  init    =$9109   ?  serve   =$910C   ?  peek    =$910F
?  poke    =$9112   ?  call    =$9115   ?  putmsg  =$9118   ?  getmsg  =$911B
?  bcast   =$911E   ?  bpoke   =$9121   ?  peekinc =$9124   ?  run     =$9127
?  brun    =$912A   ?  rcvctl  =$912D   ?  rcvptr  =$9130   ?  RARL=>AL=$9133
?  rcvlong =$9136      @       =$9138      self    =$9139      sbuf    =$913A
   rbuf    =$9142     locaddr =$914A     retrylim=$914C      servecnt=$914D
   arbxv   =$914E     tolim   =$914F     reqctr  =$9150      reqretry=$9151
   retrycnt=$9152     errprot =$9153     ckerr   =$9155      frmcerr =$9157
?  nadaver =$9159     idtable =$915A     instald =$917A      nparms  =$917B
   errstop =$917C     varcmd  =$917D     vartype =$917E      varadr  =$917F
   cmdtable=$9181     INSTALL =$921C     AMPNADA =$9247      cmd     =$9254
   chain   =$925D     comp    =$9260     timeout =$9364      null    =$936E
   idtbl   =$9371     service =$937D     vermsg  =$9395      INIT    =$93A8
   setid   =$93DA     svrxkbd =$93F4     SERVER  =$93F7   V? ]PROTERR=$9462
   BOOTREQ =$948E     BCASTREQ=$9497   V? ]doit   =$9499      GETIDSRV=$94B2
   PEEKREQ =$94E1     PEEKSRV =$9512     PKINCREQ=$9535      PKINCSRV=$9548
   RUNREQ  =$9561     BRUNREQ =$9565     POKEREQ =$9569      setreq  =$956B
   sethooks=$958D     RUNSRV  =$9591     BRUNSRV =$95A6      POKESRV =$95A6
   rts     =$9600     BPOKEREQ=$9601     BPOKESRV=$960C      CALLREQ =$961B
   CALLSRV =$962E     docall  =$9631     REQUEST =$963A      RCVDACK =$9696
   PUTMREQ =$96BF     GETMREQ =$96E1     BCASTARB=$9712   MV ]dly    =$9717
MV ]delay  =$9719     ARBTRATE=$9800     SENDACK =$9812      SENDRSP =$9814
   SENDCTL =$9826     SENDPKT =$982C     lasl=>al=$98E9      la=>a   =$98F3
   RCVCTL  =$9900     RCVPKT  =$9906     RCVPTR  =$990A      PROTERR =$998F
   rarl=>al=$9998     ra=>a   =$99A2     DSENDLNG=$99AD      SENDLONG=$99B0
   RCVLONG =$99CF   V  ]end    =$99F9   MD align   =$8000   MD dlyms   =$8000
MD delay   =$8000   MD mov16   =$8000   MD inc16   =$8000      keybd   =$C000
   kbstrobe=$C010   ?  VBL     =$C019   ?  spkr    =$C030      an0     =$C058
   an1     =$C05A     dsend   =$C05A   ?  an2     =$C05C   ?  an3     =$C05E
   pb0     =$C061     pb1     =$C062      drecv   =$C062   ?  pb2     =$C063
?  ptrig   =$C070     dsk6off =$C0E8      zipslow =$C0E8      ERROR   =$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
```