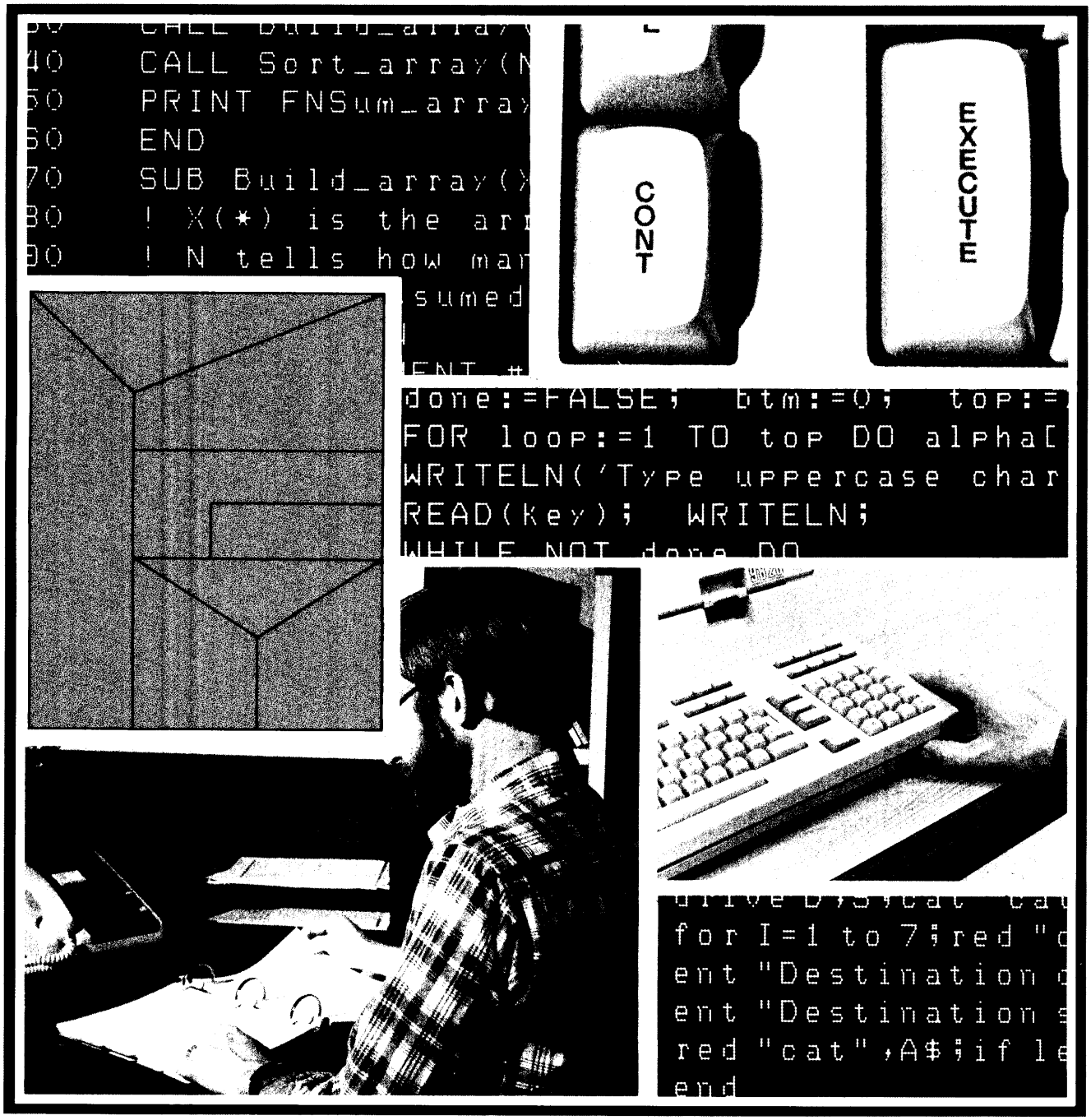
HP Series 200 Computers

HEWLETT PACKARD

Pascal
Source Code Listings



Warranty Statement

Hewlett-Packard makes no expressed or implied warranty of any kind, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose, with regard to the program material contained herein. Hewlett-Packard shall not be liable for incidental or consequential damages in connection with, or arising out of, the furnishing, performance or use of this program material.

HP warrants that its software and firmware designated by HP for use with a CPU will execute its programming instructions when properly installed on that CPU. HP does not warrant that the operation of the CPU, software, or firmware will be uninterrupted or error free.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Software clause in DAR 7-104.9(a).

Pascal 2.0
Source Code Listings
for Series 200 Computers

Volume I

Manual Part No. 09826-90074

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

March 1983...First Edition

Table of Contents

Pascal Procedure Name Cross-Reference

This section of the of the System Internals Documentation contains a cross-reference of the Pascal procedure names used in the Pascal source listings.

```
actual_tfr_type
     kernel          451   465
act_tfr
     srm_drv        1195  1342
     kernel          465
     iolib          1136  1287
ac_chars
     srmdam          542   589
ac_char_arr
     srmdam          540   542
ac_createlink
     packets         186
     srmdam          178   188   292   540   586   587
ac_execute
     packets         187
ac_generic09
     packets         188
ac_generic1
     packets         188
ac_generic10
     packets         189
ac_generic11
     packets         190
ac_generic12
     packets         191
ac_generic13
     packets         192
ac_generic14
     packets         193
ac_generic15
     packets         194
ac_generic16
     packets         195
ac_generic17
     packets         188
ac_generic18
     packets         189
ac_generic19
     packets         190
ac_generic2
     packets         189
ac_generic20
     packets         191
ac_generic21
     packets         192
ac_generic22
     packets         193
ac_generic23
     packets         194
ac_generic24
     packets         195
ac_generic25
     packets         196
ac_generic3
     packets         190
ac_generic4
     packets         191
ac_generic5
     packets         192
ac_generic6
     packets         193
ac_generic7
     packets         194
ac_generic8
     packets         195
ac_manager
     packets         181
     srmdam          178   188   292   540   587  1005  1039  1072
ac_purgelink
     packets         185
     srmdam         1005  1039  1073
ac_read
     packets         182
ac_search
     packets         184
ac_write
     packets         183
add
     initload       1726  1763  1763  1764  1770
     librarian      1569  1574  1577  1586  1588  1605
addit
     loader          865
     librarian       184  1606  1697  2454  2458  2483
addr
     mini            432   481   509
     bootdam         692
     loader         1120  1480  1487  1518  1618  1655
     isr              90    94   119   123
     misc            267   431   432   443
     ldr             870   872
     kbd             339   351
     keys            129   145   175   195
     crt             142   143   324   334   344   359   375
     ctable          608   642   872   885   886   887   898   900
     convert          64   117
     srm_drv        1367  1374  1375  1376  1420  1458  1553  1591  1629  1630  1630  1740  1742  1742  1790  1791  1791  1908
                    1969
     srmdam          465   466   502   561   780   812   895   978  1049  1054  1055  1063  1087  1088  1093  1098  1099  1107
                    1110  1111  1164  1205  1197  1198  1219  1257  1322  1346  1347  1359  1360  1375  1400  1414  1421  1458
                    1575
     srmam            69    77   116   124   195   202
     librarian       129   135  2812  2824
     hminit          168
     qminit          557   584   609   636   668   671   714   718   755   780  1032
     xminit         1349  1659
     miui            166   486   488   490   521
     tapebkup         56    83   116   139   169   213   807   808   876   965   976  1057  1095
     heapt           139
     kernel          812  1123  1160  1161
     h_drv           217   235   243
     dl_drv          217   229   237
     dma_drv         211   215   228   234
     g_drv           216   228   236
     dc_drv          247   255   265   273   572   582   590
```

```
     xminit          1236  1420  1425  1427  1429  1430  1432
badfile
     globals           47
     misc             685   686
     lifdam           405
badio
     lifdam            86   103   171   177   184   216   297   378   469   489   490   863   877  1045  1048  1051  1053  1066
                      1142
badverify
     f9885             49
bad_blocks
     packets          346
bad_track
     hminit           300   323   326   331   333
bat
     bat               32    95
     clock             33
bat8041cmdreg
     bat               44    73
bat8041datareg
     bat               45    67    85
bat8041statusreg
     bat               43    51    59
batbytereceived
     bat               38    82    85
     clock            195   196   200   201   202
batcommand
     bat               37    64    90
     clock            143   144   194   195   196   200   201   202
batdataout
     bat               66    74    75    76    77    78
batinit
     bat               39    88    98
batstatus
     bat               48    51    52    56    59    60
batterypresent
     bat               35    70
     clock            188
bba
     tapebkup         486
     cs80             212
bcb
     h_drv            136
     di_drv           137
     g_drv            139
     dc_drv           184   338   346
     rs_drv           140
bcd
     bootdam          555   556
     lifdam            37    41
     miui             383   384
     cs80             712   841   946   979
bcd12
     miui             384   398   407   420
bcd_product_number
     cs80             709   838   841   943   976   979
bdattype
     srmdam            66   911  1018
bdx_chain_semaphore
     dischpib          52   470   482   485
bdx_nb
     dischpib          54   445   497   498   500   500   501   513   518
bdx_pre_eoi
     dischpib          53   514   515   517   518
     amigo            813   941
     cs80            1270  1307
bdx_proc
     dischpib          55   446   526
beep
     kbd              126   219   233   302
     keys             250   270   415
     crt              390
     librarian         89  1550  3468  3470  3473  3475  3477  3479  3480  3483  3484  3487  3489  3491  3493  3497  3499  3501
                      3503  3505  3507  3510
beeper
     kbd              127   221   282
bell
     misc             162
bellchar
     misc             162   369
     crt              390
     librarian         91    96
     printer           46
bep
     printer           44   214   221
bia_ptr
     dischpib          78    86    87    89   100   111
bia_type
     dischpib          75    78
biggraphics
     globals          285
     crt              169   184
     kernel           929
bigsize
     heapt             71   103   112
bigtemp
     kernel           878  1063  1064
bigtemp3
     kernel           879  1008  1010  1070  1071
binand
     kernel           614
     iolib           2176  2197  2504  2522  2573  2638
bincmp
     kernel           620
     iolib           2176  2197
bineor
     kernel           618
binior
     kernel           616
     iolib           2113  2134
bip
     dischpib          94   100   107   111   114   115   116
bip_type
     hminit           258
```

```
     librarian      2806  2840  2841  2842  2842  2868  2870  2877  2879  2888  2891
     xminit         1157  1158
     miui            447   457   482   486   486   488   490
     printer          40    49   299
     f9885            40   110   281
     amigo           431   623
     cs80            900  1111
buffered_transfer
     dischpib         45
     amigo           679   684   685   721
buffersize
     crt             162   174   180   187
buffer_busy
     iolib          1114  1525  1529  1530
buffer_data
     iolib          1100  1152  1157  1186  1251  1277  1405  1428  1490
buffer_reset
     iolib          1097  1163  1187  1251
buffer_space
     iolib          1098  1181  1188  1263  1360  1408  1455  1511
buffsize
     misc            465   472   484   486   497   502   506   507   516   568   568   570   576   576   579   583   583   590
                     606   615   616   618   654
buflength
     ascii            10    21    22   290
bufparams
     ascii            18    49    85   291   320   330
bufptr
     lifdam          372   387   428   431
     librarian      2806  2842  2844  2844  2848  2848  2851  2851  2863  2864  2864  2865  2865  2883  2884  2886  2887
     xminit         1215  1310  1373  1459  1553  1588  1629
     f9885            82   178   213   213   281   282
     cs80            913   915  1122  1134  1150  1151  1151  1159  1166  1179  1180  1180  1189  1190  1190  1199  1222  1237
bufsize
     globals          69
     bootdam         545   748   757   764   773
     misc            523   527   528   528   529   536   541   542   542   544   544   545   548
     initunits       714   723   727   731   803
     ascii             7    14   182   190   195   202   219   222   226   239   245   248
     srmam            42   180   195   202
buf_empty
     srm_drv        1198  1267  1279  1374  1419  1447
     kernel          474
     dc_drv          375   377   377
     iolib          1139  1157  1170  1442  1444  1444
     dischpib        463
buf_fill
     srm_drv        1199  1267  1278  1291  1376  1420  1448
     kernel          475
     dc_drv          394   396   396   406   406   424   425   425   428   442   442
     iolib          1140  1157  1169  1188  1469  1471  1471
     dischpib        464
buf_info
     dischpib         56   448   492   502   510
buf_info_type
     srm_drv        1124  1264  1271  1284
     kernel          462
     dc_drv          339
     iolib          1064  1069  1075  1081  1088  1093  1095  1097  1098  1100  1102  1104  1107  1111  1114  1124  1152  1163
                    1181  1195  1221  1306  1330  1356  1399  1424  1451  1479  1507  1525
     dischpib         56
buf_ptr
     srm_drv        1194  1278  1279  1291  1372  1417  1445
     kernel          472
     iolib          1134  1139  1140  1169  1170  1188
     dischpib        461
buf_read_cmd
     amigo            58   724
buf_size
     srm_drv        1197  1291  1373  1418  1446
     kernel          473
     iolib          1138  1188
     dischpib        462
buf_type
     kernel          460   472
buf_write_cmd
     amigo            59   725
buildchar
     keys             64   316   399   399   405
buildcount
     keys             65   315   400   400   401
burst_tfr
     kernel          454
busaddr
     xminit         1160  1164  1171  1175
busy
     kbd              89   256
     crt             163
     bat              52
bus_address
     ctable          971  1124  1126
     printer          71   115   246   294
bus_addressed
     printer          73   238   248   268   300
bx_bufptr
     dischpib         49
     amigo           623   766   839   842   950   950
     cs80           1237  1282
bx_length
     dischpib         50
     amigo           624   747   748   951   951   954
     cs80           1238  1269  1271  1282
bx_strt_rcrd
     dischpib         48
     amigo           622   688   689   738   814   826   839   862   917   949   949   958   959
     cs80           1236  1269
bx_tfr_length
     dischpib         51
     amigo           748   749   766   842   863   864   917   949   950   951
bx_tries
     dischpib         47
     amigo           621   828   828   830   849   856   867   903   903   904
byte
     globals          29   146   147   148   149   206   207
```

```
check
     ctable          249  446  459  460  461  468  469  470  479  480  482  491  492  493  495  502  503  504
                     513  514  515  516  525  527  536  537  538  547  549  559  560  561  562  576  577  578
                     579
check2
     lifdam          720  734  741  761
checkentry
     lifdam          786  858
checkftitle
     lifdam          214  651  950 1046 1067 1144
checkinref
     librarian      2349 2711
checkrev
     loader         1085 1578 1623
     ldr            1012 1030
checkspace
     lifdam          717  791  837  874
checktextbuf
     librarian      2311 2732 2739 2746 2752 2759 2780
checkvolid
     lifdam          295  377  482  544  573  651  949 1046 1065
check_dsj
     amigo           548  776  782
check_ioresult
     miui            373  487  489  491  527
check_qstat
     cs80            925 1273 1293 1299
check_status
     hminit          264  322  395
check_tapebuf
     cs80            998 1008 1026
chr
     mini            387
     bootdam         779  787
     loader         1289 1290
     initload       1687
     misc            159  160  162  163  164  165  166  167  168  170  171  172  251  548  585
     initunits       788
     kbd             150  151  152  152  153  154  154  155  156  156  157  158  159  160  160  184  185  186
                     187  188  189  190  192  193  204  205  206  206  207  208  208  209  210  210  211  212
                     213  214  214  263  270  278
     keys            176  180  190  199  266  266  268  268  293  294  295  297  298  321  322  338  341  344
                     345  361  362  368  380  405  414  436
     crt             124  165  196  197  304  308  335  345  355  395
     bat              67   73
     ascii           108  108  115  116  134  134  225  242  242  295  331  331
     lifdam          556
     convert          97
     srm_drv        1363 1364 1365 1366 1369 1371 1415
     srmdam          197  199
     librarian       502  503 1478 1535 3465
     miui             68   69   70   71
     tapebkup        769
     kernel          144  145  199  200  201  202  203  204  205  206  207  208  209  210  211  212  832 1128
                    1186
     dc_drv          428
     iolib           422  424  499  501 1694 1723 1764 1787 1835 1843
     printer          62   63   64  115  116  245  246
     dischpib        266  267  268  307  308  324  325  326  386
     amigo           466  470
ci
     librarian        27
cic
     cs80            325  331  332  333  333
cinfo
     misc            198
     lifdam          556  556  600  600
     srmdam          390  392  586  589  591  593  593  594  594  598  598  599  599
cinfo0
     crt             102  442
cinfo1
     crt             102
cinfo2
     crt             102
cinfo3
     crt             102
ci_clr
     cs80            328  332
ci_switch
     tail             38
ckind
     misc            183
     lifdam          591
     srmdam          577
cl
     qminit          624  632  633  634  635  636  636
clastdate
     misc            191
     lifdam          554  598
     srmdam          388  411  582
clasttime
     misc            192
     lifdam          554  598
     srmdam          389  411  582
cleandir
     lifdam          343  379  953  964 1071 1079
cleanup
     lifdam          206 1138 1151 1154
clear
     crt             245  391  392  393
     librarian      3350 3357 3360 3368 3414 3422 3427 3433 3439 3442 3448 3450
     iolib          1647 1705
clearanychar
     keys            252  329  388  394  404
cleariohook
     globals         273
     init           1117
     ctable         1098
     kernel         1165
clearline
     kbd              56  193
clearlogs
     qminit          628  634
clearscr
```

```
     misc            167
     crt             393
     tapebkup        963  1046  1081  1134  1152
clearscreen
     kbd              55   192
clearunit
     globals          66
     mini            496
     keys            155
     crt             331
     ctable          605
     srmam           187   205
     miui            490
     printer         315
     f9885           268
     amigo           575
     cs80           1077
clear_and_escape
     f9885            94   132   188   202   209   225   245
clear_hpib
     iolib          1578  1600  1687  1688  1696  1738  1740
clear_logs
     qminit          512   615   638  1002
clear_logs_option_byte
     qminit          809   928   939  1002
clear_serial
     iolib          2076  2153
clear_unit
     mini            395   497
     printer         108   316
     f9885            85   269
     amigo           490   582
     cs80           1043  1080
clock
     clock            32   218
     lifdam           26
     librarian        27
     miui             11
clockinit
     clock            30
close
     ldr             862   863   864   865   874   882
     librarian      3006  3007  3030  3182  3197  3210  3341  3543  3559  3560
     xminit         1656
closedirectory
     globals         136
     lifdam         1134
     srmdam         1604  1649
closed_share_code
     packets         164
closef
     lifdam          971  1110
closefile
     globals         135
     bootdam         721
     loader         1391
     misc            429
     lifdam         1110
     srmdam         1612  1715
closefiles
     loader         1090  1385  1561  1647
     ldr            1064
     librarian      3074  3541  3557
closein
     librarian      3024  3073  3098  3146  3174  3538  3541  3556
closeout
     librarian      3172  3483
closepack
     srm_drv        1032  1676
     srmdam          315   900
clsize
     misc            186
     lifdam          551   593   594
     srmdam          382   579
cmd
     kbd             132   259   268   268   270   277
     bat              37    64    73
     cs80            293   339   346   349   349   354   362   366   366   512   532   544   544
cmd1
     cs80            230
cmd100
     cs80            255
cmd101
     cs80            255
cmd102
     cs80            255
cmd103
     cs80            255
cmd104
     cs80            256
cmd105
     cs80            256
cmd106
     cs80            256
cmd107
     cs80            256
cmd108
     cs80            257
cmd109
     cs80            257
cmd11
     cs80            232
cmd110
     cs80            257
cmd111
     cs80            257
cmd112
     cs80            258
cmd113
     cs80            258
cmd114
     cs80            258
cmd115
     cs80            258
```

```
cmd99
     cs80            254
cmdcold_load_read
     cs80            232
cmdcopy_data
     tapebkup        202
     cs80            232
cmddescribe
     cs80            243   430
cmdinit_diagnostic
     qminit          554
     cs80            242
cmdinit_media
     qminit          581
     tapebkup         80
     cs80            243
cmdinit_util_mem
     qminit          607   633   707
     cs80            242
cmdinit_util_rem
     cs80            242
cmdinit_util_sem
     qminit          665   706
     cs80            242
cmdlocate_and_read
     cs80            230  1267
cmdlocate_and_ver
     tapebkup        168
     cs80            231
cmdlocate_and_wrt
     cs80            230  1268
cmdno_op
     qminit          750
     tapebkup        112   165   201
     cs80            243   485   537   541
cmdrelease
     cs80            233   417
cmdrelease_denied
     cs80            233
cmdrequest_status
     cs80            233   402
cmdset_address_1v
     tapebkup        113   205   210
     cs80            234   538
cmdset_address_3v
     qminit          751
     cs80            234
cmdset_block_disp
     cs80            234
cmdset_burst_abt
     cs80            245
cmdset_burst_lbo
     cs80            245
cmdset_length
     tapebkup        166   199
     cs80            236   542
cmdset_options
     cs80            244   468
cmdset_release
     cs80            244   450
cmdset_retadd_mode
     tapebkup        110
     cs80            248
cmdset_retry_time
     cs80            244
cmdset_rps
     cs80            244   503
cmdset_status_mask
     cs80            245   486
cmdset_unit_0
     qminit          534
     tapebkup         45    54
     cs80            238   331   365   377   535
cmdset_unit_1
     cs80            238
cmdset_unit_10
     cs80            240
cmdset_unit_11
     cs80            240
cmdset_unit_12
     cs80            241
cmdset_unit_13
     cs80            241
cmdset_unit_14
     cs80            241
cmdset_unit_15
     qminit          553
     tapebkup        198
     cs80            241   449
cmdset_unit_2
     cs80            238
cmdset_unit_3
     cs80            238
cmdset_unit_4
     cs80            239
cmdset_unit_5
     cs80            239
cmdset_unit_6
     cs80            239
cmdset_unit_7
     cs80            239
cmdset_unit_8
     cs80            240
cmdset_unit_9
     cs80            240
cmdset_vol_0
     qminit          535
     tapebkup         46    55
     cs80            246   389   536
cmdset_vol_1
     cs80            246
cmdset_vol_2
     cs80            246
cmdset_vol_3
```

```
     tapebkup        902   924   926   931   933   938   939   947
crtcommand
     crt             194   208   209   219   220   439   440   444
crtcon
     kbd              65   148   202
     crt             444
crtconfigreg
     globals         284
     crt             442
crtconsttype
     kbd              39    65   149   202
crtcontroladdr
     kbd              62   144   198
     crt             434   435
crtcrec
     kbd              49    80   183
crtctrl
     kbd              80   183
crtfrec
     kbd              41    78   173
crtidreg
     crt             100   442
crtinfo
     kbd              81   195   335   339   340
     keys            446
     crt             116   424
     mminit          108
     printer         158   172
crtinit
     crt              47   422   458
crtio
     initunits       721   818   824
     keys            166
     crt              48   317   343   357   358
crtirec
     kbd              60    81   141   195
crtline
     miui            221   243   245   252
     tapebkup        902   924   926   933
crtmemaddr
     kbd              62   143   197
     crt             426
     mminit          109
     printer         160   174
crtptr
     mminit           99   109   112
crtregtype
     crt              64   194
crttype
     kbd              79   182
crtword
     kbd              92   102
     crt              36
crt_wtb
     kernel          704  1155
crunch
     globals         136
     lifdam         1131
     srmdam         1750
crunchv
     lifdam          368  1131
cs80
     qminit          466   792
     tapebkup         14   225
     cs80             60   554   885
cs80disc
     ctable          164   563   754   783   908  1022  1142  1145  1146  1149  1214  1266
cs80dsr
     qminit          792
     tapebkup        225
     cs80            551   885
cs80dvr
     cs80            882
cs80init
     cs80             29
cs80io
     cs80            899  1064
cs80ir
     qminit          463   792
cs80tape
     ctable          165   580   754   783   908  1019  1143  1144  1147
cs80tape_dav
     ctable          967  1020  1111  1148  1149  1286  1289
cs80tbdvr
     tapebkup        222   761
cs80tbr
     tapebkup         11   225   761
cs80_dav
     ctable          813   904   908
cs80_devid
     cs80            906   933   977   979   979
cs80_mp
     cs80            905   934   981
csamigo
     hminit          137   198
     amigo            34   425
cstart
     misc            188
     lifdam          548   551   595
     srmdam          384   580
csvp
     hminit          363   379   405   405   406   408   410
ct
     cs80             94
ctable
     ctable           24
cteol
     misc            164   369   369   411   411
     crt             392
     librarian        96  1531  1533  1533  2994  3147  3201  3249  3265  3271  3277  3278  3285  3303  3319  3352  3377  3385
                    3401  3402  3409  3410  3411  3419  3430  3445  3461  3463  3463  3529
     tapebkup        857  1124
cteos
     misc            166
```

```
     crt             391
     librarian      1522  1530
ctet
     amigo           242   248   251   252   253   254   255   256   257   258   259   260   261   262   263
ctr
     ctable          138   691   807   952
ctsec
     clock           162   167   168
cts_line
     kernel          300
     iolib          2257  2305
currentif
     qminit          822
     cs80            113
current_record
     packets         577
current_timeout
     printer          78   244   252   261   271   301
current_vol
     ctable          216   217   393   398   404   406
current_vu
     tapebkup        496   689
     cs80            198   696
cursaddr
     crt             214   216   217   218   219   220   423   436   437   438   439   440
cursormask
     kbd              73   162   216
cvtdatetime
     lifdam          529   554   555   598
cyl
     amigo            90   352   357   372   373
cylinder_compare_error
     amigo            72   804   817
cylinder_ert
     qminit          501
cyln
     qminit          472   513   732   752   752
cyl_per_med
     hminit          242   437
     amigo           129   193   194   196   197   198   199   200   201   202   203   225   351   352   372
d
     kbd             262   263
     bat              66    67
     lifdam          113   120   120   121   121   122   122   123   123   124   124   125   126
     librarian       410   659   704   707   713   885   920   931  1057  1065  1083  1125  1162  1181  1183  1198  1198  1202
                    1206  1206
     miui             15   178   201   316   346   468
     amigo           100
d0
     ctable          709
d1
     ctable          709   769
d10
     ctable          709
d11
     ctable          709
d12
     ctable          709
d13
     ctable          709
d14
     ctable          709
d15
     ctable          709
d16
     ctable          710
d17
     ctable          710
d18
     ctable          710
d19
     ctable          710
d2
     ctable          709
     kernel          738   743   753
d20
     ctable          710
d21
     ctable          710
d22
     ctable          710
d23
     ctable          710
d24
     ctable          710
d25
     ctable          710
d26
     ctable          710
d27
     ctable          710
d28
     ctable          710
d29
     ctable          710
d3
     ctable          709
d30
     ctable          710
d31
     ctable          710
d4
     ctable          709
d5
     ctable          709
d6
     ctable          709
d7
     ctable          709
d8
     ctable          709
d9
     ctable          709
daccess
```

```
     loader          840
     librarian      2799  2967
dam
     globals         144
     loader         1391  1610
     initload       1677
     misc            479   597
     initunits       778
     ctable          310   334   334   368   373   374   377   378   380   382   428   447   483   496   505   528   550   563
                     580
     ascii            81
     miui            246   522   526
     tapebkup        927
     lockmod          26    49    67
damrequesttype
     globals         133   140
     bootdam         543   607
     misc            211   421
     initunits       715   734   739   806
     lifdam           28   189
     srmdam           45  1581
damtype
     globals         140   144
     initunits       756
     ctable          310
dam_fib
     miui            370   518   526
dam_name
     ctable          286   323   359
dam_proc
     ctable          308   323   324   329   334
dat
     srm_drv        1099  1112  1995  2002  2081  2088  2099
data
     bootdam         780   788
     kbd             318   323
     crt             194   197
     packets         723   809
     srm_drv        2099
databyte
     kbd              96    99
datafib
     lifdam          374   388   389   424   428   431   453   453
datafile
     globals          51
     bootdam         668   673   707
     loader          837
     misc            453   689   691
     lifdam          142   169   284   525   904   941   985
     srmdam           88   921  1406
dataout
     kbd             262   271   272   273
dataoverrun
     f9885            49   241
datasize
     loader          922  1460
     librarian      2008  2510  2545  2556  2558  2631  2729  2770
datasource
     qminit          701   713
dataspace
     lifdam          661   699   722   725   727   735   737   754   762   764   776   790   791   836   837   868   869   872
                     873   874   998  1018  1024  1025  1030  1030  1031
datatype
     loader          857   922   931
     srmdam           68  1407
     librarian      2404
data_communication
     ctable          854
data_error_log
     qminit          514   642   680  1026
data_overrun
     tapebkup        449   643
     cs80            176   649
data_records
     packets         158
     srmdam         1055  1088  1099  1111  1376
date
     globals         230
     loader          941
     lifdam          529   531   532   533   534   534   535   535   535
     packets         251
     srmdam          341   342   343  1420
     librarian       102   107   327  2043
     miui             52    61    64
daterec
     globals         217   230
     loader          836   840   941   942
     misc            191
     clock            41    43    87   147
     lifdam          115   308   323   529   566
     srmdam          332   362
     librarian        38    39   102
     miui             44    52
datetimerec
     globals         229
     srmdam         1393
date_type
     packets         249   282   283   736   738
     srmdam          331
dav
     ctable           90    96    98   100   104   106   108   205   696   746   756   812   813   850   904   957   960   961
                     962   963   964   965   967   969   996   999  1065  1068
dav_line
     kernel          286
day
     globals         219
     clock           107   153
     lifdam          122   122   317   332   332   533   534   535   568
     packets         253
     srmdam          342   342   362
     librarian       109   112  3565
     miui             44    65    68    68    69    89
db
     hminit          179   187
```

```
dtr_line
     kernel          305
     iolib          2105  2125  2168  2188  2237  2285
du
     globals         148
     mini            489   492   492
     initload       1681
     initunits       782
     ctable           92    96    98   100   104   106   109   226   230   231   232   233   234   235   236   237   338   360
                     444   446   447   457   461   461   462   475   480   483   487   493   496   500   504   505   509   515
                     528   532   538   550   554   561   563   567   578   580   645   645   756   766   785   883   883   957
                    1006  1006  1006  1032  1035  1035  1036  1075  1081  1081  1088  1088  1089  1118  1173  1174  1178  1190
                    1191  1196  1197  1240  1250  1260  1268  1290  1310  1311  1330  1340  1341  1346  1347
     srm_drv        1497
     srmdam          392
     qminit          534
     miui             36    36   200   345   346
     tapebkup         45   204   209   573   575   810   810   883
     f9885           126   131   148   164   201
     amigo           268
     cs80            331   389   535
dumbuf
     srm_drv        1124  1192  1216  1303  1309  1311  1320  1321  1328  1330  1336  1338  1350  1372  1373  1374  1376  1417
                    1418  1419  1420  1445  1446  1447  1448
dummy
     misc            299   357
     keys             43   468
     crt             153
     librarian       433   449
     miui            393   428
     kernel          723   728   733   738   743   748   753   876   912   919   920   940   941   941   941   942   994   995
                     995   997   997  1037  1045  1051  1067  1068
     h_drv           180
     dI_drv          181
     dma_drv         172   191
     g_drv           183
     iolib          2093  2112  2113  2113  2114  2133  2134  2134  2135  2156  2175  2176  2176  2177  2196  2197  2197  2198
                    2220  2270  2271  2318  2319  2390  2422  2423  2424  2425  2433  2434  2435  2459  2521  2522  2522  2523
                    2543  2572  2573  2573  2574  2596  2637  2638  2638  2639
dummy1
     lifdam           46   247   479
dummy2
     lifdam           47   247   479
dummy3
     lifdam           50   247   479
dummy4
     lifdam           57   505
dummycleario
     init           1104  1117
dummydebug
     init           1103  1116
dummykbdhook
     kbd             318   349
dummykbdrelease
     keys            439   451
dummykbdwait
     keys            434   450
dummytimerhook
     kbd             323   350
dummyword
     dc_drv          194   203   206   207   288   319   321   533   608   609   610   633   634
     rs_drv          180
dummy_boolean
     ctable          975  1523  1525
dummy_byte
     hminit          161   167   168   168
dummy_char
     dischpib        260   270   318   328
dummy_driver
     kernel          718   725   730   735   740   745   750   755  1100  1101  1202
dummy_drivers
     kernel          652  1099  1123  1152
     h_drv           189
     dI_drv          190
     g_drv           192
     dc_drv          543
     rs_drv          189
dummy_driver_a
     kernel          728  1102  1104  1108  1109  1111
dummy_driver_b
     kernel          748  1112  1113
dummy_driver_ba
     kernel          753  1114
dummy_driver_c
     kernel          723  1103  1110
dummy_driver_dw
     kernel          738  1107
dummy_driver_w
     kernel          733  1105
dummy_driver_wa
     kernel          743  1106
dummy_dsj
     amigo           508   521
dummy_isc
     dc_drv          195   199   204   212   214   215   220   223   224   226
dummy_pr
     srm_drv        1202
     kernel          360  1135  1190
     dma_drv         142   159   179   182
     dc_drv          466
     iolib          1144
dummy_sl
     srm_drv        1201
     kernel          361  1134  1189
     dma_drv         180   183
     iolib          1143
dummy_tfr_1
     kernel          441
dummy_tfr_2
     kernel          445
dumpa
     crt             111   446
dumpalpha
     keys            229   308   323
```

```
extended_features_mode
     midecs          35
     qminit         930
     xminit        1333
     miui           154   155
extension
     bootdam        566   671   673
     lifdam          67   593   599   925   941   942   985   986
extentsize
     srmdam          52  1028  1056  1089  1100  1112  1273  1273  1372  1377
extg
     g_drv          110   177
exth
     h_drv          107   174
extlist
     loader        1173
extmodptr
     loader        1301  1303  1304  1304  1313
extra
     loader        1083  1136  1153  1154  1158  1159  1164  1165  1590  1602  1603  1613
     ldr            947   949   950   954
extrastates
     librarian      582   589   895   915  1186
extsize
     loader         956  1285  1285  1286  1287  1288  1291
     librarian      367   367   368   393   394   554   644  1771  1772  2066  2821
extsiztype
     librarian      545   554
exttable
     loader        1172
e_rcrd
     amigo          794   814   815   826   835   839   848   848   862   916   917   920
f
     globals        140
     bootdam        543   544   607   692   696   696   697   697   699   699   700   700   700   705   705   705   707   709
                    710   711   712   712   712   713   713   714   714   747   752   773
     loader        1118  1120  1120
     misc           211   261   265   266   266   267   267   267   270   421   426   431   432   443
     initunits      715   734   739   806   830
     kbd            329   336   337
     ctable         243   422   422   422   428   434   440   440   447   447   453   453   462   471   471   483   483   496
                    496   505   505   528   550   563   563   580   580   597   600   605
     lifdam          28   189   216   216   217   217   297   378   469   574   574   574   576   587   587   588   617   617
                    627   627   896   897   898   898   899   899   900   900   901   901   903   903   903   904   912   916
                    917   918   935   939   940   941   942   950   950   951   954   956   961   976   982   985   986   987
                   1002  1033  1036  1045  1052  1057  1066  1068  1072  1088  1090  1090  1107  1108  1110  1113  1116  1121
                   1129  1130  1132  1133  1135  1139  1142  1144  1145
     srmdam          43   117   120   131   138   140   421   427   431   451   453   472   492   494   535   555   618   643
                    684   848   853   855   874   890   927   944   946   951   957   967   984  1009  1047  1083  1121  1126
                   1132  1145  1147  1149  1172  1184  1186  1188  1207  1213  1215  1225  1233  1241  1265  1271  1284  1297
                   1310  1314  1315  1327  1339  1349  1379  1385  1393  1419  1427  1430  1451  1454  1458  1458  1474  1494
                   1579  1597  1644  1654  1657  1660  1667  1670  1679  1682  1694  1697  1706  1709  1716  1718  1720  1722
                   1726  1729  1735  1737  1739  1741  1743  1745  1747  1752
     srmam           50    60    97   107   144   149
     librarian      102   112   114   115   117   127   129   129   133   135   135   217   220   227   228   229   231   232
                    233   234  2406  2410  2810  2812  2812
     hminit         176   184   430
     miui            15   176   197   329   345   469
     heapt          133   145   146   150   156   157   159   160   162   163   164   165   167   168   174   177   179   182
                    183   184   185
     lockmod          7     8     9    14    17    26    39    42    49    55    58    67
     amigo          112   662   886   899
f0
     ctable         706
f1
     ctable         706
f2
     ctable         706
f3
     ctable         706
f4
     ctable         706
f5
     ctable         706
f6
     ctable         706
f7
     ctable         706   767
f9885dvr
     f9885           34
f9885init
     f9885           31
f9885io
     f9885           40    45
fakeaddr
     heapt          133   139   147
fakeone
     heapt           76   120   139   140   141   159
false
     mini           466   469   500
     bootdam        637   642   664   678   712   740   764   772
     loader         926  1012  1020  1320  1320  1334  1340  1341  1433  1436  1448  1458  1461  1462  1607  1623
     initload      1688  1689  1690  1691  1692  1693
     misc           274   283   438   448   531   586   656
     initunits      789   791   793   794   815   815   818   820   820   822   824   824
     ldr           1012  1025  1029
     kbd            161   174   175   176   194   215   267
     keys           194   256   284   342   358   381   453   454
     crt            443
     tail            75    86
     ctable         243   263   345   347   714  1078
     ascii           39   290
     lifdam         172   178   208   233   250   258   275   277   277   278   278   286   304   399   401   575   618   652
                    741   837   840   849   855   855   874   979  1001  1001  1048  1053  1069  1101  1104  1107
     convert         59
     srm_drv       1204  1344  1429  1439  1449
     srmdam          64   215   284   426   447   487   491   554   645   680   681   794   817   823   843   855   866   884
                    932   991  1029  1274  1302  1332  1379  1410  1412  1463  1467  1492  1505  1512  1519  1526  1595  1646
                   1650  1651  1677  1691  1706
     srmam          162
     librarian      168   177   180   211   223   224   355   355   394   521   521   532   536   608   644   665   667   668
                    827   836   954  1119  1255  1346  1377  1377  1417  1417  1519  1590  1605  1627  1638  1652  1681  1689
                   1722  1729  1772  1839  1868  1893  1919  1962  2005  2009  2009  2010  2123  2161  2256  2287  2388  2446
                   2454  2483  2490  2569  2603  2622  2627  2722  2829  2851  2876  3021  3035  3076  3111  3179  3194  3200
```

```
     misc            535   573   622   644
fextra
     globals         123
fextra2
     globals         124
ff
     misc            167
     keys            321
ffpw
     globals         101
     srmdam          126   692   833  1625  1626  1634  1763
fgetxy
     mminit          107
     miui            243   257
     tapebkup        924   938
     printer         189
fgotoxy
     librarian        96  1504  1521  1531  1533  2994  3050  3147  3201  3249  3265  3271  3277  3278  3285  3303  3319  3360
                    3385  3424  3437  3461  3463  3570  3572
     miui            245   252   258   267
     tapebkup        926   933   939   948
     printer         194   205   212
fhs_tfr
     kernel          455
fib
     globals          64    72   140
     bootdam         543   607
     loader         1118  1604
     misc            211   261   421
     initunits       715   734   739   806   830
     ldr             951
     ctable          597
     lifdam           28   189
     srmdam           43   117   131   431   472   535   618   848   874   927   957   984  1132  1172  1207  1225  1265  1284
                    1396  1398  1427  1451  1474  1579
     srmam            50    97   144
     librarian       127   133  2810  2900  3143
     miui            370   453
     lockmod           7     8     9    14    39    55
fibp
     globals          64    68    70    74
     mini            309   384
     bootdam         544   747
     loader          912
     misc            464   496   515
     initunits       713   722   726   730   802
     ldr             870   872   878   880
     keys             55   137
     crt              48   317
     ascii             6    13
     lifdam          239   244   273   303   303   336   337   374   391   509   511  1102  1124  1125
     srmdam         1190  1192  1196  1198  1202  1202
     srmam            39   177
     swvol            22    24
     librarian      2824
     printer          40    49
     f9885            40
     dischpib         43
     amigo           430
     cs80            899   912   914   998  1004  1017  1121  1221
fibptr
     srmdam         1396  1400  1414  1421
     heapt            28
     dischpib         43   139
     amigo           601
     cs80           1214  1232
fid
     globals          43   128   267
     loader         1093  1583
     ldr             848   855   925   980
     ctable          978
     lifdam          163
     srm_drv        1033  1070  1074  1085  1095  1101  1105  1108  1677  1685  1841  1851  1877  1888  1937  1949  1991  2005
                    2018  2030  2041  2050  2077  2091
     srmdam         1135  1589
     miui            125
     tapebkup        847
fid1
     srm_drv        1067  1821  1831
fid2
     srm_drv        1068  1822  1832
fidleng
     globals          20    43
     srmdam         1636
fidptr
     srmdam         1135  1155  1156  1157
fieldwidth
     miui            225   258
     tapebkup        906   939
fileblock
     loader         1029  1343  1539  1544
     librarian      1379  1382  1474  2702  2703  2841  2887
filedirectory
     loader          879  1033
filedirptr
     loader          879   910  1053
filefib
     loader         1028  1342  1437  1438  1538  1543
     librarian      1379  1382  1473  2328  2369  2841  2887
fileid
     globals         107
     mini            514
     bootdam         710   711
     misc            266   437
     lifdam          241   279   392   627   896
     srmdam          307   309   310   312   312   315   317   904  1240  1274  1302  1304  1307  1335  1344  1366  1381  1382
                    1384  1404  1417  1432  1438  1463  1465  1631  1760
     srmam            67    75    84   114   122   131
     miui            481
     f9885           286
     amigo           622
     cs80           1111
fileinfopack
     srm_drv        1069  1840
```

```
      loader          858
      librarian       178  1650  1915  2450  2459  2861
generalvalue
      loader          871   920  1458
      librarian       166   168  1255  1626  1627  1675  1681  1681  1721  1722  2005  2256  2388  2626  2627  2717  2722  2829
                     2848  2851  2876
general_0
      kernel          635  1244
      h_drv           173
      di_drv          174
      dma_drv         135
      g_drv           176
      dc_drv          190   526
      rs_drv          174
      iolib           149   317  1913  2086  2384
general_1
      iolib           117   695  1914
general_2
      iolib           650
general_3
      iolib           959
general_4
      iolib          1048
german
      keys            39   351   371
getbounds
      librarian      1501  1545  1546
getbytes
      loader         1084  1111  1235  1259  1262  1286  1320  1322  1328  1433  1456  1458  1464  1515  1542  1602  1604
      ldr             949   951
getcodeblocks
      librarian      1352  1496
getcommand
      librarian      3455  3592
getcursor
      globals          67
      crt             328
getdeftable
      loader         1246  1333
getexttable
      loader         1281  1345
getinstrbytes
      librarian       474   488   525   814   988
getinstruction
      librarian       485  1272  1288  1313
getioerrmsg
      misc            207   298   387
      ldr             988
      librarian      3517
      miui             26   143
      tapebkup        579   790   865
getpaoc
      srmdam          637   700   743   751   760
getrecsize
      ascii           164   195   310
getsdate
      lifdam          308  1129
getsize
      heapt            97   104   163   169   242
getspace
      lifdam          657   912
getuntildelim
      srmdam          192   225   240   250
getvolumedate
      globals         133
      lifdam         1129
      srmdam         1747
getvolumename
      globals         133
      bootdam         692
      misc            431
      lifdam         1114
      srmdam         1745
      miui            246   522
      tapebkup        927
getxy
      crt             223   328
get_amigo_letter_proc
      ctable          843   892   927
get_bootdevice_parms
      ctable          696   746  1135
get_cs80_letter_proc
      ctable          844   891   928
get_cs80_parms
      ctable          813   904  1215  1289
get_letter
      amigo           428   439
      cs80            895   938
get_letter_proc
      ctable          866   870   872
get_letter_type
      ctable          823   831
get_map
      hminit          238   435
      amigo           140   189   223   344   368
get_message
      kernel          202
      iolib          1850
gfiles
      globals         257
      ldr             864   865   878   879   880   882   883
      init           1097  1098  1099  1100  1118
      crt             129   187   190
      swvol            22    24
gheight
      crt             149   156   179
gheightb
      crt             149   157   173
gl
      ctable          831   872
global
      loader          858
      librarian       177  1649  1700
globalbase
```

```
     loader          953  1412
     librarian       365  1768  2054
globaldelta
     loader         1027  1412
     librarian      1578  1579  1768
globalsize
     loader          952  1379  1379  1413  1413
     librarian       366  1769  1769  2053  2687
globase
     loader         1025  1412
     librarian      1768  2687
gmem
     crt             158   186
gmemb
     crt             159   185
gmembtype
     crt             157   159
gmemtype
     crt             156   158
go
     initload       1710  1791
     ldr             854   895
     tail             75    88
goff
     crt             137   143
gon
     crt             136   142
goodio
     lifdam           82   289   429   432  1091
gotoxy
     crt             238   327   365   445
gp
     f9885            37
gpio
     f9885           107
gpioclear
     f9885            90    97
gpiodmain
     f9885           170
gpiodmaout
     f9885           175
gpiodma_proc
     f9885           107   170   175   178
gpiotype
     f9885            80   107
gpiowordin
     f9885           129   196
gpiowordout
     f9885           124   127   128   146   151   163
gpio_card
     kernel          224  1014
gpio_drivers
     g_drv           165   191   192   216
gpio_enable
     f9885            72   178
gpio_enable_type
     f9885            66    73
gpio_initialize
     g_drv           101
gpio_r3_type
     f9885            66    74    76   108
gptr
     librarian      1563  1675  1676  1677  1690  1711
     f9885            80    87    90    97   121   122   124   127   128   129   146   151   163   178   183   191   196   263
graphicsbase
     crt             160   185   186
graphicstate
     kbd             114
     keys            219   225
     crt             141   141   142
graphics_card
     kernel          226  1040
gread
     bootdam         581   632   655   686   710   740   741   743   750   799
gringo
     keys             39   287   462   469
growbyte
     crt             153   186
gsize
     heapt           134   169   170   170
gtl_message
     kernel          199
     iolib          1734
gvalue
     loader         1104
gvaluestring
     librarian        41   154   155   155   159   177   187   191   192   205   209   212   212   356   537  1120  1389  1418
                    2291  2570  2571  2634  2635  2645  2646
gvp
     loader          898  1459  1526
     ctable          277
     librarian       139   162   200   355   360   403   404   406   521   532  1363  1376  1377  1417  1624  1676  1676  1677
                    1690  1719  1894  1895  2006  2024  2171  2252  2254  2287  2383  2444  2510  2521  2545  2556  2558  2568
                    2601  2615  2621  2691  2713  2727  2785  2824  2844  2872
gvptr
     loader         1104
     librarian      1716  1719  1721  1722
gvrbase
     librarian      1242  1251  1252  1255
gvrequal
     librarian      1888  1893  1924  1926  1960  2504
gvrp
     librarian      2200  2568  2569  2621  2622
gvrptr
     loader          871   898  1272
     librarian        87   139   149   166   168   172   195   300  1252  1891  2134  2200
gvrstring
     librarian       139   355   521   532   536  1119  1377  1417  2287  2569  2622
gwidth
     crt             150   155   180   186
gwidthb
     crt             150   151   157   174   185
gword
     crt             152   155
```

```
     tapebkup        1122
hour
     globals          224
     clock            124   125   167
     lifdam           123   123   536   569
     srmdam           348   349   363
     librarian        231
     miui              78    80
hp7905
     ctable           160   518   753   783  1022  1210  1245  1254  1415  1422
     amigo             44   168   179   185   504   957
hp7905_mp
     ctable           189   519  1210  1409
hp7906
     ctable           161   522   753   783  1022  1211  1245  1434  1439
     amigo             44   169   179   185   504
hp7906_mp
     ctable           190   522  1211  1428
hp7920
     ctable           162   543   753  1022  1212  1238  1451
     amigo             44   170   179   504
hp7920_mp
     ctable           191   543  1212  1445
hp7925
     ctable           163   544   753  1022  1213  1238  1463
     amigo             44   171   179   191   504   957
hp7925_mp
     ctable           192   544  1213  1457
hp8290x
     ctable           156   505   752  1004  1010  1165  1302  1524
     hminit           440
     amigo             44   164   191   320   520   683
hp8290x_dav
     ctable           960  1011  1107  1171  1308
hp8290x_default_dav
     ctable            95  1107
hp913x_a
     ctable           157   540   752  1022  1207  1238  1379
     hminit           441
     amigo             44   165   687   957
hp913x_a_mp
     ctable           186   540  1207  1373
hp913x_b
     ctable           158   541   753  1022  1208  1238  1391
     amigo             44   166
hp913x_b_mp
     ctable           187   541  1208  1385
hp913x_c
     ctable           159   542   753  1022  1209  1238  1403
     hminit           441
     amigo             44   167   492   687   957
hp913x_c_mp
     ctable           188   542  1209  1397
hp98620
     kernel           239
     dma_drv          193
hp98622
     kernel           248  1015
     g_drv            213   249
     f9885            262
hp98623
     kernel           249
hp98624
     kernel           246  1001
     h_drv            213   259
hp98625
     kernel           250  1021
     di_drv           214   253
     dischpib         229   467   494
hp98626
     kernel           247  1007
     rs_drv           207   238
     iolib           2119  2182  2276  2430  2494  2569  2622  2673  2708
hp98627
     kernel           252  1039
hp98628_async
     kernel           251
     dc_drv           206   321   561   609
     iolib           2099  2162  2228  2419  2465  2561  2602  2666  2701
hp98628_dsndl
     kernel           236
     dc_drv           207   562   610
hp98629
     srmam            150
     kernel           237
     dc_drv           223   563   616
hp9885
     ctable           154   483   752   781  1004  1013
hp9885_dav
     ctable           962  1014  1104  1194  1344
hp9885_default_dav
     ctable            99  1104
hp9885_mp
     ctable           184   482   483
hp9895
     ctable           155   496   752   990  1004  1028  1034  1206  1234
     hminit           440
     amigo             44   163   208   316   492   657
hp9895_dav
     ctable           961   991   992  1039  1048  1108  1187  1337
hp9895_default_dav
     ctable            97  1108
hp9895_mp
     ctable           185   495   496  1206
hphydata
     hminit           203   233   245
     miui             468
hpibaddr
     xminit          1194  1414  1457  1551  1562  1586  1627
hpibamigo_clear
     dischpib         196   297
     amigo            474   524
hpibamigo_identify
     qminit           858
```

```
     printer          182   269
infost
     librarian         56   388  1423  1445  1489  1498
infostart
     librarian         83  2807  2817  2895  3020  3042  3075  3193
initbat
     bat               30
initbootdam
     bootdam          542   730
     initload        1674
     initunits        736
initclock
     clock             45   210   221
initcrt
     crt               30
initdiag
     qminit           548   554
initdumbuf
     srm_drv         1190  1220  1361  1403
initfilekinds
     misc             212   665
     init            1123
initfiles
     init            1095  1143
initfkinds
     initload        1701  1780
initialize_bkgnd
     miui             537
     dischpib          61    82   540
initialize_media
     qminit           510   563   586   986
     tapebkup          25    62    85   739
initialize_medium
     miui             280   542
initialize_options_byte
     qminit           807   926   935   948   986
     tapebkup         236   724   739
initial_seek
     amigo            543   625   652
initiate_data_transfer
     amigo            546   753   754   760
     cs80             923  1272  1278
initiate_diagnostic
     qminit           509   539   559   980
initiate_transfer
     dischpib         421   484   488
initisrib
     xminit          1204  1349  1659
initkbd
     kbd               30
initkeys
     keys              57   443   482
initlink
     librarian       3039  3487
initload
     initload           2
initloader
     loader          1094  1653
     initload        1781
initmedia
     qminit           575   581
     tapebkup          74    80
initsyscom
     kbd              328   354
initsysisr
     initunits        745   832
initsysunit
     ldr              850   887
     init            1138
     ctable           625
inittime
     clock            173   212
initunitable
     initunits        809   835
initunitentry
     initunits        755   814   817   819   821   823
initunits
     initunits        707
     init            1088
     xminit          1188
initutil
     qminit           602   607   627   633   659   665   695   706   707
initvects
     init            1110
init_913x_abc
     hminit           359   441
init_9895_913x_chinook
     hminit           282   440
init_bkgnd
     dischpib          26
init_cmd_buf
     hminit           160   165
init_dc
     dc_drv           499   648
init_discint
     di_drv           150   264
init_dma
     dma_drv          109   242
init_d_cmd
     hminit           165
     amigo             56
init_gpio
     g_drv            149   260
init_hpib
     h_drv            149   270
init_root_password
     packets          616
init_rs
     rs_drv           147   251
init_scanstuff
     ctable           811   919  1099
init_srm
     srmdam            26
init_tapebuf
```

```
     cs80              45    49  1327
init_with_d_bits
     hminit           141   158   343   397
inoaccess
     globals          194
     misc             347
     srm_drv         1161
inodirectory
     globals          192
     misc             337
     lifdam           261   475
     srmdam           808
     miui             120
     tapebkup         843
inoerror
     globals          184
     mini             377   400   455   521
     bootdam          690   754
     loader          1121
     misc             302   503   530   546   554   639
     kbd              337   338
     keys             144
     crt              130   188   323
     ctable           984
     lifdam            83   246   261   474   476  1103
     convert           51    53
     srm_drv         1132  1146  1535  1537
     srmdam            79   220   258   454   495   500   504   563   706   729   732   745   763   771   776   782   786   801
                      814   827   856   896   948  1013  1052  1057  1067  1081  1084  1092  1095  1105  1108  1117  1148  1150
                     1161  1187  1189  1191  1193  1216  1275  1309  1316  1325  1326  1341  1345  1348  1378  1380  1415  1418
                     1433  1439  1456  1460  1464  1466  1501  1503  1552  1593  1645  1658  1661  1668  1671  1680  1683  1695
                     1698  1707  1727  1757
     srmam             73    82    88    91   120   129   135   138   156   158   183
     librarian        130   136   286   288
     mminit           118
     hminit           260   451
     qminit           677   724   878  1106
     miui             119   375
     tapebkup         268   289   322   362   365   400   406   470   493   498   529   571   581   599   606   664   686   691
                      717   749   842
     lockmod           16    28    41    50    57    69
     f9885            180   186   187   293
     dischpib         118
     amigo            571   579   595   632   646   775
     cs80             579   586   678   693   698  1055  1069  1082  1092  1098  1104  1105  1216  1246  1292
inofile
     globals          186
     bootdam          716
     misc             312
     lifdam           653   966  1048
     srm_drv         1179
     srmdam          1050  1090  1324
inoroom
     globals          185
     misc             310
     lifdam           489   490   877
     srm_drv         1173
inotclosed
     globals          186
     misc             314
     srm_drv         1171
inotdirect
     globals          189
     misc             328
inotlockable
     globals          193
     misc             342
     srmdam           893
     lockmod           19    44    60
inotondir
     globals          195
     misc             350
     srm_drv         1164
     srmdam           899
inotopen
     globals          186
     misc             315
     srm_drv         1153
inotreadable
     globals          189
     misc             326
inotvalidsize
     globals          189
     misc             325
     srm_drv         1137
inotwriteable
     globals          189
     misc             327
inounit
     globals          185
     bootdam          719
     misc             311
     srmdam           859
     miui             143
     tapebkup         865
inpass
     srmdam          1484  1492  1508  1512  1519  1527
inpdtable
     xminit          1241  1492  1498  1499
input
     init              30
     ldr              862   870   871
     swvol              3
     librarian         24
     miui               8
     tapebkup           8
     kernel           100   692
     h_drv            100
     di_drv           100
     dma_drv           99
     g_drv            101
     dc_drv           102
     rs_drv           103
inrefindex
```

```
     di_drv          191   255
     g_drv           193   251
     dc_drv          544
     rs_drv          190   241
     iolib           174  1204
     printer          96   137
     dischpib        220
iod_isr
     kernel          337  1101
     h_drv           191   238
     di_drv          192   232
     g_drv           194   231
     dc_drv          545   585
     rs_drv          191   223
iod_ppoll
     kernel          347  1111
     h_drv           201
     di_drv          202
     iolib          1944
     dischpib        251   270   328   412
iod_rdb
     kernel          338  1102  1154
     h_drv           192
     di_drv          193
     g_drv           195
     dc_drv          546
     rs_drv          192
     iolib           183   734   736   742   744   817   848   928   949
     dischpib        353   359   382   385
iod_rds
     kernel          342   844  1106
     h_drv           196
     di_drv          197
     g_drv           199
     dc_drv          550
     rs_drv          196
iod_rdw
     kernel          340  1104
     h_drv           194
     di_drv          195
     g_drv           197
     dc_drv          548
     rs_drv          194
     iolib           206
iod_send
     kernel          346  1110
     h_drv           200
     di_drv          201
     iolib           325
     printer         113   115   116   242   245   246   255
     dischpib        264   266   267   268   291   308   309   322   324   325   326   336   360   386   525
iod_set
     kernel          348  1112
     h_drv           202
     di_drv          203
     g_drv           202
     iolib          1593  2139
     dischpib        289   306
iod_temp
     kernel          566   581   584   589   689   704
iod_test
     kernel          350  1114
     h_drv           204
     di_drv          205
     g_drv           204
     iolib          1617  2324
iod_tfr
     srm_drv        1350
     kernel          345  1109
     h_drv           199
     di_drv          200
     g_drv           201
     dc_drv          552
     rs_drv          198
     iolib          1317  1342  1388  1416
     dischpib        502
iod_wtb
     kernel          339  1103  1155
     h_drv           193
     di_drv          194
     g_drv           196
     dc_drv          547
     rs_drv          193
     iolib           194   794   870
     printer         250
     dischpib        285   290   307
iod_wtc
     kernel          343   860  1107
     h_drv           197
     di_drv          198
     g_drv           200
     dc_drv          551
     rs_drv          197
iod_wtw
     kernel          341  1105
     h_drv           195
     di_drv          196
     g_drv           198
     dc_drv          549
     rs_drv          195
     iolib           220
ioerror
     librarian        98  3158  3241
     iolib           974   978   983   984   986  1015  1016  1018  1040
ioerror_message
     iolib           974   978  1042
ioescapecode
     srm_drv        1380  1463
     srmdam         1771
     srmam           215
     kernel          151   577
     dc_drv          401   433   626
     printer          98   102   139
     dischpib        217
```

```
ioe_bad_cnt
     srm_drv        1316
     kernel          163
     iolib           998  1241
ioe_bad_sct
     kernel          173
     iolib          1008  2097  2109  2129  2160  2172  2192  2226  2267  2315
ioe_bad_tfr
     kernel          160
     dc_drv          359
     iolib           995  1366  1381
ioe_bad_tmo
     kernel          164
     iolib           233   238   999
ioe_buf_busy
     srm_drv        1275  1318
     kernel          162
     iolib           997  1174  1248  1440  1467
ioe_crd_dwn
     kernel          174
     iolib          1009
ioe_dc_act
     kernel          186
     iolib          1028
ioe_dc_car
     kernel          185
     iolib          1027
ioe_dc_clk
     kernel          183
     iolib          1025
ioe_dc_conf
     kernel          188
     iolib          1030
ioe_dc_conn
     kernel          187
     iolib          1029
ioe_dc_cts
     kernel          184
     iolib          1026
ioe_dc_fail
     kernel          180
     iolib          1015  1022
ioe_dc_ovfl
     kernel          182
     iolib          1024
ioe_dc_reg
     kernel          189
     iolib          1031
ioe_dc_rval
     kernel          190
     iolib          1016  1032
ioe_dc_usart
     kernel          181
     iolib          1023
ioe_eod_seen
     kernel          175
     dc_drv          402   434
     iolib          1010
ioe_isc
     kernel          503   575
     dc_drv          403   435   627
     printer          92    98   100   102   139
     dschpib         217
ioe_isc_busy
     srm_drv        1327  1335
     kernel          161
     iolib           996  1261  1275
ioe_misc
     kernel          176
     dc_drv          328
     iolib           421   498   983  1011  1487  1983  2417  2423  2434  2442  2463  2483  2505  2514  2527  2547  2556  2579
                    2600  2613  2633  2644  2664  2681  2699  2716
ioe_not_act
     kernel          156
     iolib           585   635   991  1931
ioe_not_dvc
     kernel          157
     iolib           992  1783
ioe_not_hpib
     kernel          155
     iolib           344   349   428   505   990  1987
ioe_not_lstn
     kernel          169
     iolib          1004
ioe_not_sctl
     kernel          171
     iolib          1006  1817  1824
ioe_not_talk
     kernel          168
     iolib          1003
ioe_no_card
     kernel          154   770   789   809   828
     iolib           989
ioe_no_data
     srm_drv        1337
     kernel          159
     iolib           994  1280  1431  1493
ioe_no_dma
     kernel          166
     iolib          1001
ioe_no_driver
     srm_drv        1314
     kernel          165   720
     iolib          1000  1231
ioe_no_error
     kernel          153
     iolib           984   988
ioe_no_space
     srm_drv        1329
     kernel          158
     iolib           993  1266  1458  1514
ioe_no_word
     kernel          167
     iolib          1002
```

```
ioe_rds_wtc
     kernel          172
     iolib          1007
ioe_result
     srm_drv        1380  1463
     kernel          502   576
     dc_drv          402   434   637
     iolib          1036
     printer          93   101   104
     dischpib        218
ioe_sr_fail
     kernel          191
     iolib          1036
ioe_sr_toomany
     kernel          179
     iolib          1020
ioe_timeout
     srm_drv        1380  1463
     kernel          170
     iolib           462   539  1005
     printer         104
     dischpib        218
ioinitialize
     iolib           132   153
iolibrary_kernel
     kernel          100
iomaxisc
     kernel          140   260   261   946  1120  1179  1199
     h_drv           210   257
     di_drv          211   251
     g_drv           210   247
     dc_drv          558   599
     rs_drv          204   236
     iolib           405   482   568   617  1227  1709  1731  1759  1798  1813
iominisc
     kernel          139   260   261   946  1120  1179  1199
     h_drv           210   257
     di_drv          211   251
     g_drv           210   247
     dc_drv          558   599
     rs_drv          204   236
ior
     mini            414   454   455   464   468   472
     loader         1591  1646  1649
     kbd             277
     dischpib         65   155   158
ioread_byte
     kernel          662   799   812   994  1053  1053  1056
     h_drv           225
     di_drv          220
     g_drv           223
     dc_drv          220   577   613
     rs_drv          215
ioread_word
     kernel          656   761   774
     dma_drv         145   162   191
iores
     hminit          259   260
     qminit          675   677   722   724
     miui             23    26
     tapebkup        571   579   581   787   790
     dischpib         39   118   133   158
     amigo           775
     cs80           1292
ioresc
     mini            311   322   401   468   472   490   493   508   511   513   518   521
     mminit          118
     printer         105   132   140   224   308   330
     f9885            89   119   138   156   180   217   220   224   228   235   236   237   242   262   280   283   285   290
                     293
     cs80           1010  1133
ioresc_bkgnd
     hminit          275   277   311   329   345   350   355   376   399   410   424   425   430   431   432   442
     qminit          859  1083
     tapebkup        249   280   303   346   404   499   509   604   692   703   735
     dischpib         65   155   218   221   237   337
     amigo           172   212   470   477   478   483   516   519   528   529   533   535   605   607   609   666   669   800
                     824   869   870   876   879   881   888   890   891   892   896   898   902   905   909   913   921   922
                     925   930   942
     cs80            584   699   843   847   974  1053  1055  1241  1308
ioreset
     iolib           134   171
ioresult
     mini            324   483
     bootdam         621   690   716   719   720   724   726   754   758   760   765   768
     loader         1121  1139  1646  1649
     misc            231   238   268   284   291   425   457   474   480   487   503   530   546   554   593   598   639   658
     initunits       804   807
     ldr             914   988  1063
     kbd             337   338
     keys            144   205
     crt             130   188   323   418
     tail             89
     ctable          606   984
     ascii            27    34    38    45    51    82   137   250   285   334
     lifdam           83    87   208   246   261   261   303   337   453   474   475   476   653   956   966  1088  1090  1103
                     1125  1148
     convert          51    53    53
     srm_drv        1132  1135  1136  1137  1138  1141  1143  1146  1151  1153  1159  1161  1164  1168  1171  1173  1174  1177
                     1179  1180  1181  1182  1183  1185  1381  1383  1459  1464  1466  1535  1537
     srmdam             79    80   160   162   220   258   391   393   400   454   495   500   504   563   658   665   667   672
                      706   729   732   737   745   756   763   771   776   782   786   801   814   827   856   896   948  1013
                     1050  1052  1057  1067  1081  1084  1090  1092  1095  1096  1103  1105  1108  1116  1117  1148  1150  1161
                     1187  1189  1191  1193  1216  1242  1275  1303  1309  1316  1324  1325  1326  1341  1345  1348  1358  1361
                     1378  1380  1415  1418  1433  1439  1456  1460  1464  1466  1501  1503  1552  1593  1599  1645  1658  1661
                     1668  1671  1680  1683  1695  1698  1707  1727  1757
     srmam            73    82    88    91   120   129   135   138   151   156   158   159   183   188   193   200   211   216
     librarian        96   130   136   221   286   288  2813  3010  3183  3212  3219  3342  3581
     hminit          450
     qminit          875   878  1105
     miui            142   357   375   376
     tapebkup        267   288   321   361   365   528   716   748   864  1031  1066  1104
     heapt            32
     lockmod          16    19    22    28    33    41    44    47    50    57    60    63    69
```

```
     iolib           760   761
     printer         334
     f9885           100   186   186   187   188   255
     dischpib        137   138   140   170
     amigo           567   571   579   595   637   644
     cs80           1069  1073  1081  1092  1104  1107  1109  1115  1246
ioresult_value
     mini            311   322   324
iores_value
     f9885            94   100
iorsltwd
     globals         178
     mini            311   313   322   329   335   414
     bootdam         594
     lifdam           86
     tapebkup        392   393   588   589
     dischpib         39    65   155
     cs80            568   569
iorval
     mini            313   329   377   380   400   401   454
     mminit          118   118
iorval_to_report
     tapebkup        393   400   493   494   498   499   588   599   686   687   691   692
     cs80            568   579   693   694   698   699
iostatus
     kernel          668   838   848
     iolib           361   376   452   460   461   529   537   538  1997  2007  2016  2026  2112  2133  2175  2196  2270  2318
                    2504  2521  2572  2637
ios_access_to_file_not_allowed
     srm_errs        959
     srm_drv        1161
ios_attach_table_full
     srm_errs        943
     srm_drv        1136
ios_bad_file_name
     srm_errs        962
     srm_drv        1140
ios_bad_select_code
     srm_errs        937
     srm_drv        1135
ios_conflicting_share_modes
     srm_errs        961
     srm_drv        1167
ios_corrupt_directory
     srm_errs        952
ios_deadlock_detected
     srm_errs        977
     srm_drv        1166
ios_device_drivers_dont_match
     srm_errs        941
ios_directory_formats_dont_match
     srm_errs        945
ios_directory_not_empty
     srm_errs        971
     srm_drv        1182
ios_ds_rom_missing
     srm_errs        939
ios_duplicate_filenames
     srm_errs        965
     srm_drv        1174
ios_duplicate_passwords
     srm_errs        976
     srm_drv        1159
ios_eof_encountered
     srm_errs        981
     srm_drv        1177
ios_error_base
     srm_errs        935
ios_error_top
     srm_errs        987
ios_file_in_use
     srm_errs        963
     srm_drv        1170
ios_file_locked_please_retry
     srm_errs        983
     srm_drv        1168
ios_file_not_directory
     srm_errs        970
     srm_drv        1181
ios_file_not_found
     srm_errs        968
     srm_drv        1179
ios_file_pathname_missing
     srm_errs        950
     srm_drv        1141
ios_file_unopened
     srm_errs        955
     srm_drv        1153
ios_illegal_byte_number
     srm_errs        951
     srm_drv        1143
ios_improper_mass_storage_device
     srm_errs        944
ios_insufficient_disk_space
     srm_errs        964
     srm_drv        1173
ios_invalid_file_code
     srm_errs        982
     srm_drv        1183
ios_invalid_file_id
     srm_errs        947
     srm_drv        1138
ios_invalid_file_size
     srm_errs        946
     srm_drv        1137
ios_invalid_ios_request
     srm_errs        942
ios_invalid_protect_code
     srm_errs        973
     srm_drv        1157
ios_link_to_directory_not_allowed
     srm_errs        978
     srm_drv        1164
ios_no_capability_for_file
```

```
     srm_errs        967
     srm_drv        1156
ios_no_reply
     srm_errs        984
     srm_drv        1146
ios_password_not_allowed
     srm_errs        958
     srm_drv        1155
ios_password_not_found
     srm_errs        975
     srm_drv        1158
ios_phys_eof_encountered
     srm_errs        966
     srm_drv        1176
ios_purge_on_open
     srm_errs        985
     srm_drv        1171
ios_rename_across_volumes
     srm_errs        979
ios_software_bug
     srm_errs        936
ios_successful_completion
     srm_errs        953
     srm_drv        1145
ios_system_down
     srm_errs        954
     srm_drv        1148
ios_unallocated_extent
     srm_errs        938
ios_unsupported_dam
     srm_errs        940
ios_unsupported_directory_operation
     srm_errs        960
     srm_drv        1163
ios_volume_down
     srm_errs        980
     srm_drv        1151
ios_volume_in_use
     srm_errs        969
     srm_drv        1180
ios_volume_io_error
     srm_errs        949
ios_volume_labels_dont_match
     srm_errs        957
ios_volume_not_found
     srm_errs        972
     srm_drv        1150
ios_volume_offline
     srm_errs        956
     srm_drv        1149
ios_volume_recoverable_error
     srm_errs        948
ios_volume_unrecoverable_error
     srm_errs        974
iouninitialize
     iolib           133   162
iounit
     xminit         1160  1164  1171  1175
iowrite_byte
     kernel          665   818
     dc_drv          224   226   618   620
iowrite_word
     kernel          659   780
io_bit
     kernel          262   326   328   748   753
     h_drv           139   140   141
     di_drv          140   141   142
     g_drv           140   141   142
io_byte
     kernel          263   392   406   420   464   488   490   566   581   664   667   801   820
     h_drv           181
     di_drv          182
     g_drv           184
     dc_drv          534
     rs_drv          181
io_carriage_rtn
     kernel          145   695
     iolib           821   898   911
io_drv_ptr
     srm_drv        1230  1350
     kernel          431   844   860  1123  1160  1161  1202  1204
     h_drv           217   219   261
     di_drv          217   219   255
     g_drv           216   218   251
     dc_drv          572   585
     rs_drv          210   223   241
     iolib           174   183   194   206   220   325   390   724   789   810   841   866   924   945  1204  1317  1342  1388
                     1416  1593  1604  1617  1944  2139  2202  2324
     printer          96   111   137   234
     dischpib        220   249   262   281   304   320   336   349   380   410   502   510
io_error_link
     kernel          550   894
     iolib          1040
io_escape
     srm_drv        1275  1314  1316  1318  1327  1329  1335  1337
     kernel          564   572   720   770   789   809   828
     dc_drv          214   328   359
     iolib           233   238   344   349   421   428   462   498   505   539   585   635  1174  1231  1241  1248  1261  1266
                     1275  1280  1366  1381  1431  1440  1458  1467  1487  1493  1514  1783  1817  1824  1931  1983  1987  2097
                     2109  2129  2160  2172  2192  2226  2267  2315  2417  2423  2434  2442  2463  2483  2505  2514  2527  2547
                     2556  2579  2600  2613  2633  2644  2664  2681  2699  2716
io_find_isc
     kernel          566   581   586   590   720
     dc_drv          199   320   328
io_funny_proc
     kernel          355   386   400   414   476   487   489
io_init_dc
     dc_drv          517   530   653
io_init_discint
     di_drv          167   178   269
io_init_dma
     dma_drv         128   171   247
io_init_gpio
     g_drv           167   180   265
```

```
     kernel          497
     iolib           750   837   885   919   928   929   941   949  1956  1960  1963
io_work_str
     iolib           701   729   734   735   736   737   742   743   744   745   749   750   754   785   792   794
io_work_word
     dma_drv         140   145   157   162
ip
     srm_drv        1357  1367  1368  1375  1376
ipart
     clock            51    73    78    79    82   139   140   193   197   199   203
ipointer
     globals          31
iptr
     librarian      1570  1581  1582  1584  1584  1585  1589  1589  1598  1604  1611  1612  1614  1615  1616  1616
iramigo
     hminit          134   198
ires
     librarian        58    96    99  3517  3523  3533
isc
     iolib          1116  1534  1537
isc_busy
     iolib          1116  1534  1539  1540
isc_table
     srm_drv        1229  1255  1303
     srmam           149
     miui            345
     kernel          508   766   785   805   824   843   859   946  1123  1125  1126  1127  1160  1161  1180  1200  1217  1222
     h_drv           211   258
     dl_drv          212   252
     dma_drv         149   166   193   221
     g_drv           211   248
     dc_drv          204   223   320   559   600
     rs_drv          205   237
     iolib           173   182   193   205   219   241   324   335   337   359   374   389   409   438   486   515   575   625
                     724   725   789   790   810   811   841   842   866   867   924   925   945   946  1203  1230  1258  1269
                    1272  1283  1286  1316  1318  1341  1343  1387  1389  1415  1417  1537  1592  1603  1616  1943  1979  2095
                    2099  2158  2162  2224  2228  2415  2419  2430  2461  2465  2494  2545  2561  2569  2598  2602  2622  2662
                    2666  2697  2701
     printer         293
     f9885           260
     dischpib        219   228   235   249   279   302   335   347   378   410   430   492   510
isc_table_type
     kernel          430   509
     printer          69
     dischpib        258   276   299   316   343   369   426
isc_te
     dischpib        258   262   316   320
isc_te_ptr
     dischpib        276   279   280   281   299   302   303   304   343   347   348   349   369   378   379   380   426   430
                     436   441   448
isr
     isr              32
     kbd              34
     xminit         1189
     h_drv           172
     dl_drv          173
     dma_drv         134
     g_drv           175
     dc_drv          189   525
     rs_drv          173
isrchange
     isr              55   136
isrib
     globals         201   203
     kbd             138
     xminit         1204
     kernel          385   399   413   491   492
isribp
     isr              43    50    53    56    65    67    75    83    98   107   108   112   121   137   144
isribptr
     globals         202
     initunits       716   741
     kbd             287
isrlink
     isr              38    60
     xminit         1349
isrmcatchall
     globals         195
     misc            352
     srm_drv        1185  1383  1466
     srmdam         1772
     srmam           216
isrproctype
     globals         202   209
     isr              38    45    55    60    78   136
     kernel          337  1101
isrunlink
     isr              52   111
     xminit         1659
     h_drv           234
     dl_drv          228
     dma_drv         210   214
     g_drv           227
     dc_drv          581
     rs_drv          219
issue_cmd
     hminit          155   165   189
     amigo           144   240   300   310   334   388   401   412   751
issue_command_message
     cs80            922  1255  1260
issue_transfer_request
     amigo           545   703   708
istrovfl
     globals         190
     misc            330
     iolib           761
itet
     amigo           447   452   457   458   459   460   461   462   463
itoomanyopen
     globals         194
     misc            346
     srm_drv        1136
iu
     cs80             92
```

```
j
     bootdam          614   695   696   697   699   700
     initload        1736  1746  1750  1750  1750  1752
     crt              162   185   185   185   186   186   186
     ctable           970  1511  1512  1512  1512  1512  1514  1515
     ascii            101   112   114   114   115   116
     convert           46    66    68    90   110   113   113   114   115   117   120
     srmdam           442   482   548
     librarian        105   113   114   114   783   821   824   830   833  1452  3188
     xminit          1273  1276  1277  1293  1309  1310  1310
     miui             371   512
     heapt            215
jlang
     keys              44   287   288   355   462   469   469
     crt              313
jmpstates
     librarian        857   960   966
jnum
     keys              43   468
jptr
     librarian       1570  1596  1598  1600  1601  1602
junkint
     librarian        313   349   350   350   354   355  1353  1361  1362  1362
k
     crt              423   444   444   444
     clock             88    92    93
     lifdam           100   104   105   345   349   349   352   359   362   362   363
     librarian        467   469   470   783   823   824   832   833
     xminit          1282  1296
     miui              15   182   190   201   330   347   470
k1
     clock             88    93    94    95
k2
     keys             105   106   110   113
     clock             88    96    97    98
kanaalphabet
     keys             109   368
kanaalphabettype
     keys             106   109
kanaflag
     keys              53   287   358   359   360   454
kanatocrtlookup
     crt               72   308
kanatocrtlookuptype
     crt               61    72
katakana
     keys              39   287   355   367   413   469
     crt              313
kbd
     kbd               32   360
     keys              33
     crt               33
     bat               33
     clock             33
     tail              32
     mminit            61
     tapebkup         761
     printer           37
kbd8041cmdreg
     kbd              166   270   278
kbd8041datareg
     kbd              110   263   295
     keys             467   473
kbd8041statusreg
     kbd              111   255   294
     keys             430
kbdcommand
     kbd              132   259   284   353
     keys             456   457   458   465   471
     clock             74    75    76    77    80    81   141   142
     tail              87    92
     mminit           104   113   117
kbddata
     kbd              289   295   300   308   313
     keys             209   266   268   290   291   318   334   339   348   351   352   352   353   354   358   359   360   360
                      362   366   368   370   380   387   444   467   468   473   474
kbdhook
     kbd              116   313   349
     keys             257   313   314   452
kbdhooktype
     kbd               96   116
     keys              66
kbdinit
     kbd              129   344   363
kbdio
     initunits        725   820
     keys              55   137
     crt              330   338   354
kbdisr
     kbd              287   351
kbdisrib
     kbd              138   351
kbdlangjumper
     keys              42   287   288   355   460
     crt              313
kbdreg
     kbd              137   277   277   278   348
kbdreleasehook
     keys              47   275   420   451
kbdstatus
     kbd              252   255   256   260   288   294   297   300   313
     keys             209   264   281   285   286   289   318   334   339   427   430   431
kbdwaithook
     keys              46   184   450
kbd_crt_drivers
     kernel           651  1152  1153  1160  1161
kbd_rdb
     kernel           689  1154
kbeepdur
     kbd              124   221   347
kbeepfreq
     kbd              123   221   346
kbl
     keys             140   157   158
```

```
     srm_drv        1301  1306  1309  1311  1315  1328  1336  1347
lcursaddr
     crt             201   205   206   207   208   209
ldr
     ldr             842
     init           1088
     tail             32
     ctable          148   952
     librarian        27
left
     kbd              66   151   205
leftchar
     misc            163
     crt             381
leftover
     loader         1209  1320  1322  1328  1330
len
     loader         1225  1236  1237  1237  1237  1238  1238  1238  1239  1240  1251  1270  1271  1271  1271  1272  1272  1272
                    1273  1274  1400  1423  1455  1456  1456  1457
     librarian      1242  1249  1250  1250  1250  1251  1258  1258  1400  1413  1414  1415  1415  1738  1805  1865  1866  1867
                    1880  1935  1984  1985  1986  2086  2133  2134  2135  2155  2156  2157  2158
     tapebkup        160   167
     heapt           215   218   219   221   224   228   235   235   235   243   252   253   253   253
     cs80            293   512   543
length
     mini            310   385   421   433   433   436   441   445   512   512
     bootdam         588   648   655   673   674
     loader         1219  1230
     keys             56   138   166   178   190   190   190   191
     crt              49   318   330   336   339   342   342   352   355   356   356   373   375   375   395
     tapebkup         28   145   167   187   200
     printer          41    50   323   325   327   327
     f9885            41   110   284   284   286
     amigo           431   608   608   615   624
     cs80            531   543   900   913   915  1108  1108  1111  1122  1134  1141  1141  1153  1153  1156  1159  1160  1163
                    1166  1182  1182  1185  1192  1192  1195  1199  1222  1238
lentitle
     srmdam          624   646   655   687   694   707   719   725   732   746   753   755   764
len_hibyte
     packets         303
len_lobyte
     packets         302
letter
     globals         155
     initload       1687
     misc            280   280   286
     initunits       788
     ctable          344   601   646   646   823  1513
     miui             37    37   169   314   318   346
     tapebkup        811   811   879
     amigo           162   428   439   450   457   458   459   460   461   462   463   466   469   469   470   472   472   480
                     484
     cs80            845   846   852   895   938   970   972  1129  1212
letter_table
     ctable          750   774
letter_table_type
     ctable          748   751
level
     packets         304
leveltype
     globals         200
lf
     misc            165
     keys            266   294
lfb
     qminit          834
     tapebkup        115   207   212   486   971   986
     cs80             86   540   853   985
lfrag
     heapt           215   228   229   243
libfound
     loader         1074  1338
     ldr            1025  1033
lif
     ctable           48    51   372   376
lifdam
     lifdam           28   189
lifmodule
     lifdam           25  1165
lifname
     bootdam         553   559   585   651
     lifdam           36    60    90    99   131   160   614
lifnametostr
     lifdam          131   183   590
liftofkind
     lifdam          521   591   935
lifvol
     lifdam          237   253   269   310   327   472  1116  1119
lif_dam
     ctable          200   379  1290
lif_directory
     miui            416   474
lif_dir_entry
     miui            401   416   436
lif_vol_label
     miui            415   463   465
lif_vol_type
     miui            387   415   423
lim
     librarian      1401  1410  1411
line
     crt             115   118   125   128   129
     miui            220   240   254   254   255   259
     tapebkup        901   921   935   936   940
     kernel          326   328
     h_drv           139   140   141
     dl_drv          140   141   142
     g_drv           140   141   142
     iolib          1577  1579  1581  1590  1595  1601  1606  1612  1619  2075  2077  2079  2091  2101  2121  2141  2154  2164
                    2184  2204  2217  2230  2278  2326
linedel
     kbd              69   158   212
linefeed
     printer          62   320
```

```
     loader          807
     initload       1664
     ldr             844
     init           1088
     kbd             360
     keys            479
     crt             455
     bat              95
     clock           218
     tail             32
     ctable          148
     ascii           342
     lifdam         1165
     srm_drv          31
     librarian        27
     kernel         1245
     h_drv           271
     di_drv          265
     dma_drv         243
     g_drv           261
     dc_drv          649
     rs_drv          252
     f9885           309
     dischpib        537
     amigo           979
     cs80           1324
loadfib
     loader         1054  1141  1145  1154  1164  1263  1288  1325  1329  1342  1387  1390  1391  1392  1392  1393  1393  1595
                    1603  1604  1605  1610
     ldr             950   951   952   953  1001
     librarian      2908  2913  2913  2937  3028  3030  3031  3031  3032  3032  3085  3165  3165  3590
loadgvr
     librarian        87   161  1253  1376
loadinfo
     loader         1082  1206  1623
     ldr            1012  1029
     librarian       389  2818  2915
loading
     loader         1587  1618
loadq
     loader         1093  1583
     initload       1765  1771  1789
loadrom
     ldr             855   925
     tail             85
loadtext
     loader         1088  1494  1637
     ldr            1048
load_tapebuf
     cs80           1017  1149  1165  1177  1197
local
     iolib          1650  1728
local_index
     dischpib         84    88    89    96    99   100
local_lockout
     iolib          1651  1747
local_printer_dav
     ctable          963  1054  1106  1180
local_printer_default_dav
     ctable          105  1106
local_printer_timeout
     ctable           84  1181
locateandverify
     tapebkup        161   168
locate_and_verify
     tapebkup         28   145   171   519
location
     librarian       508   510   513   516   529
lock
     lockmod           7    14    29    32
lockable
     srmdam          171   215   269   880   892   909   910   942   947   951  1006  1029  1121  1126
lockdown
     misc            214   224
     clock            83
     lifdam         1156  1158
     srmdam         1775  1779
     srmam           219   223
     miui            353
     tapebkup       1027  1062  1100
     f9885           296
     dischpib        134
lockedaction
     kbd             130   224
     keys            232   238   330   335   336
locked_out
     iolib          1904  2022  2026
lockfile
     globals         138
     srmdam         1616  1735
     lockmod          26    67
lockfiles
     ldr             851   860
     ctable          616
locklevel
     globals         250
     misc            221   221   228   228   229
     init           1115
     kbd             228
     keys            334
lockmodule
     lockmod           2
lockpack
     srm_drv        1073  1876
     srmdam         1432
lockup
     misc            213   219
     clock            72
     lifdam         1097
     srmdam         1594
     srmam           184
     miui            337
     tapebkup        995  1053  1090
     f9885           258
```

```
      iolib           216   218   222
n
      loader         1038  1225  1229  1234  1234  1242  1243
      keys            241   244
      lifdam         1040  1051  1051  1052
      srmdam          134   161   365   392
      librarian       649   651  1823  1841  1847  1847  1847  1850  1873  1874  1874  3350  3352  3352  3353
      miui             15   171   326   344   468
      dischpib         72    75    88    99   113
name
      ldr             855   925   929   931   936
namelen
      srmdam         1479  1487  1520  1539  1539  1546
name_set_array
      srm_types       927   928
      srmdam          635   876
name_set_array_three
      srm_types       926
      srmdam          132   996  1231  1288
name_type
      packets         230   271   275   290   291   316   337   338   341   349   410   440   441   442   457   458   605   607
                      608   614   616   791
      srm_drv        1009  1017  1025  1029  1030  1031  1045  1059  1064  1081  1093  1511  1547  1585  1623  1654  1655  1656
                     1728  1780  1785  1902  1965
      srmdam           95   109   437   478   544   623   637   878   936   964   995  1137  1175  1176  1210  1228  1268  1287
name_type_len
      packets          52   230
      srmdam          646  1499  1546  1548
nb
      dischpib        195   197   201   274   283   287   287   341   351   357   357   424   445   462
      amigo           246   251   252   253   254   255   256   257   258   259   260   261   262   263   269
nbb
      cs80             99
nbpb
      qminit          834
      tapebkup        971   986
      cs80             98   845   846   853   969   971   985   986
ndac_line
      kernel          285
ndfs
      kbd              50   185
ndrives
      ctable          731   742
needcap
      srmdam          177   229   244   249   253
needmodeorpass
      srmdam          177   216   223
needpass
      srmdam          177   234   239   255
new
      initload       1705
      misc            671   672
      initunits       812
      init           1097  1098  1099  1100
      kbd             332
      ctable          588
      lifdam          388  1162
      srm_drv        1215  1216
      xminit         1373
      heapt            44   283
      kernel          979  1002  1008  1016  1022  1026  1042  1063  1070
      dc_drv          632
      iolib          1126  1134
      dischpib         87
      cs80             52
newbytes
      bootdam         743
      librarian      2222  2379  2396  2397  2398  2399  2400  2716  2721  2724  2725  2777  2778  2779
newdeftable
      librarian      2093  2117  2184  2188
newdirectory
      librarian        86  2073  2181  2597  2600  2950  2953  3057
neweof
      srmdam         1269  1273  1274  1277
newextsize
      librarian        85  1833  1872  1880  1880  1884  1884  2066  2067
newexttable
      librarian        84   191   192   392  1832  2068
newindex
      librarian      2218  2241  2251  2269  2286  2286  2287  2597  2598  2598  2598  2599  2601  2601  2601
newlistname
      librarian      2989  2995  2996  2997  2999  3003  3008
newmodname
      librarian        82  1780  1780  1978  1979  1980  1984  1984  1986  2964  3042  3068  3069  3069  3305  3309  3310  3408
                     3409
newmodptr
      loader         1421  1433  1434  1445  1451
newmods
      loader         1059  1198  1303  1339  1339  1360  1365  1372  1404  1427  1520  1580  1600
      ldr             962  1006
      librarian       190   386   387   390   397  1245  1246  1358  1404  1422  1432  1444  1465  1488  1497  1638  1742  1747
                     1750  1813  1998  2098  2604  2819  2896  3076  3294  3294  3373  3382  3472  3485  3492  3541  3542  3586
newname
      srm_drv        1030  1655  1667
newnfns
      srm_drv        1060  1781  1794  1807
newnsaptr
      srm_drv        1061  1782  1791
newparm
      qminit          901   910   911   911   913
newpass
      srm_drv        1031  1656  1668
newpath
      srm_drv        1062  1783  1809
newptr
      librarian      1941  1951  1959  1960  2012  2013  2214  2382  2383  2388  2396  2398  2401  2444  2474  2482  2504  2507
                     2521  2545  2555  2556  2558  2651  2652  2655  2712  2717  2722  2727
newrtpass
      srm_drv        1064  1785  1810
newstartdefs
      loader         1091  1474  1479  1490  1491
newstrptr
      librarian      1809  1866  1867  1870  2090  2156  2157  2164
newtextrec
```

```
nokeyboard
     globals          285
nonabortive_ioresult_set
     miui             119   208
     tapebkup         842   889
none
     srmdam           176
     librarian       3356  3370  3378  3408  3427  3442
nop
     qminit           744   750
     tapebkup         104   112   158   165   188   201
     cs80             480   485
nop1
     cs80             526   537
nop2
     cs80             529   541
nopassyet
     srmdam          1485  1493  1518  1524  1526
nopatch
     librarian       2438  2489  2545
nopower
     f9885              48   216
norange
     librarian        302  1276  1324  1543  1544
norecord
     f9885              48   230   235
nores
     librarian        139   190
normal_completion
     amigo              69
notcancels
     librarian       1572  1581  1588  1594
notdone
     librarian       1571  1581  1582  1590
notice
     loader           945
     librarian        329   330  2047
notrack
     f9885              49   227
notready
     f9885              62   208
notype
     librarian        295  1519
not_a_913x
     ctable          1000  1034  1036  1037
not_ready
     tapebkup         421
     cs80             157   611
nounit
     initunits        713   802   815
nowopen
     lifdam          1085  1110  1111  1112
no_card
     kernel           220   976
     printer          308
no_data_found
     tapebkup         423
     cs80             159   615
no_id
     kernel           244   948
no_isc
     srm_drv         1196  1274  1275  1289  1316  1317  1318
     kernel           193   586  1215  1220
     dma_drv          181   184
     dc_drv           463
     iolib           1137  1167  1174  1187  1200  1241  1245  1248  1431  1436  1440  1458  1463  1467  1487  1493  1514  1529
     dischpib         451
no_parity
     kernel           293
     iolib           2607  2627
no_spares_available
     cs80             156   609
no_tfr
     srm_drv         1195  1342
     kernel           451
     iolib           1136  1287
nps
     srm_drv         1026  1046  1624  1633  1645  1729  1741  1745  1766
     srmdam           172   217   275   275   276   285   439   462   466   940   947  1001  1029  1030  1030  1032  1032  1033
                     1037  1038  1055  1088  1099  1111
nrecords
     f9885              58   165
nrfd_line
     kernel           284
nsa
     srmdam           132   140   146   151   156   635   734   780   812   828   876   895   996  1009  1047  1049  1053  1054
                     1062  1063  1083  1087  1093  1098  1107  1110  1120  1121  1126  1231  1241  1256  1257
nsa1
     srmdam          1288  1314  1346
nsaptr
     srm_drv         1006  1014  1022  1042  1078  1090  1544  1553  1582  1591  1620  1629  1725  1740  1899  1908  1962  1969
nsectors
     hminit           140   146   154
nspm
     midecs            29
     mminit            86
     hminit           241
     qminit           830
     miui             432
nspt
     midecs            30
     mminit            87
     hminit           243
     qminit           833   834
     miui             432
nstates
     librarian        418   487   885  1023  1058  1079  1092  1111  1112
nta
     cs80             205
ntps
     midecs            28
     mminit            85
     hminit           242
     qminit           831
```

```
     miui            432
nul
     librarian      3189  3206
null
     misc            159
nullchar
     misc            159
     convert          82   111   114   118
     librarian      1479
null_bcd12
     miui            420   433   441
null_dav
     ctable          756   758
null_mp
     ctable          193   906   914  1219
num
     mini            391   492
     srm_drv        1508  1515
     iolib           140   202   209   665   679   700   754   880   884
number
     crt             245   249   252   252
numberofspares
     xminit         1240  1372  1474  1577
number_conversion_ok
     miui            289   299   300   305   308
number_of_tracks
     misc            263   265
number_vols
     ctable          216   217   393   397   398   399   404   406
numblocks
     librarian      2902  2927  2931  2933  2935  2935  2941  2941
numbuilt
     iolib           705   722   756   771
numdata
     kbd             132   259   271   272   273
     bat              37    64    74    75    76    77    78
numeric
     iolib           707   712   714   737   745
numfiles
     librarian      3082  3087  3088
num_bits
     iolib          2366  2457  2468  2473  2478  2497  2502  2509
num_char_bit
     iolib          2369  2541  2549
num_file_name_sets
     packets         267
     srm_drv        1515
num_protect_code_sets
     packets         424   503   606
     srm_drv        1645  1766
nvols
     ctable          973  1224  1225  1225  1226  1226  1227  1239  1241  1243  1248  1249  1251  1253  1256  1256  1259  1261
                    1263  1267  1269  1271  1375  1376  1378  1380  1387  1388  1390  1392  1399  1400  1402  1404  1411  1412
                    1414  1416  1417  1419  1421  1423  1430  1431  1433  1435  1436  1438  1440  1447  1448  1450  1452  1459
                    1460  1462  1464  1469  1475  1477  1479
nxtlink
     isr               85    89    92    94    95    95   104   114   118   121   121   123   124   124   127   129   130   132
ob
     cs80            465   469
obj
     loader         1118  1120
     librarian       127   129   133   135  2810  2812
object
     loader         1108  1501  1531  1534  1535  1536  1536  1539  1545  1546  1549
     librarian      2207  2249  2260  2305  2305  2314  2320  2320  2411  2441  2443  2506  2525  2561  2577  2635  2636  2637
                    2637  2638  2638  2639  2639  2646  2647  2648  2648  2649  2650  2650  2662  2665  2665  2666  2697  2724
                    2724  2733  2734  2735  2740  2741  2742  2747  2747  2753  2754  2755  2760  2761  2762  2777  2779  2779
                    2808  2843  2847  2847  2850  2850  2856
     heapt             44    45   288   293   297   309   312
object0
     loader         1501  1545  1549
objectaddr
     librarian      2429  2441  2443  2502  2516
obsize
     librarian      2311  2314
oc
     amigo           245   251   252   253   254   255   256   257   258   259   260   261   262   263   267   278   286   287
occurrences
     qminit          489
oct_10000
     miui            392   427
odd
     mini            510
     loader         1113  1271  1378  1379  1411  1413  1442  1455  1525
     keys            285   286   289   474
     ctable          273  1006
     ascii           107   131   168   241
     librarian       350   526   699  1134  1192  1207  1212  1250  1362  1415  1661  1767  1769  1785  1984  2023  2029  2133
                    2155  2170  2599  2690  2823  2901  3144  3289  3309
     f9885           282
oddbytefiller
     packets         299
odd_parity
     kernel          294
     iolib          2608  2628
off
     srm_drv        1098  1111  1994  2010  2080  2096
offline
     globals         156
     mini            482
     initload       1688
     initunits       789
     ctable          345   606   627
     srmdam         1598
     srmam           187
     f9885           255
     amigo           566
     cs80           1072
offset
     packets         716   806
     srm_drv        2010  2096
     srmam            57    63    67    71    75    84   104   110   114   118   122   131
     librarian       508   520   521   526   527   528   529   529   532   536  1888  1914
     qminit          698   710
```

```
     srm_types       930   931
     srmdam          173   449   941  1003
protect_code_set
     packets         314
     srm_types       930
     srmdam         1295
prtdvr
     printer          32
prtio
     printer          40    49
psa
     srmdam          173   276   285   449   457   466   941   947  1003  1029  1033  1040  1055  1088  1099  1111
psaptr
     srm_drv        1027  1047  1625  1630  1730  1742
pt
     srm_drv        1510  1518
ptp
     loader          913
     librarian      1655  1774  1775
ptr
     loader         1226  1235  1235  1568  1570  1570  1571  1573  1574  1574
     ctable          259   268   269   271   272   272   272   273   273   273   274   277   277   277
     librarian      1451  1475  1477  1479  1480  1482  1482  1483  1945  1992  1993  2032  2033  2806  2823  2824  2829  2832
                    2832  2835  2871  2871  2872  2876  2880  2880
     heapt           284   296   297   303   309   310
ptr1
     loader         1422  1431  1432  1453  1455  1455  1456  1457  1457  1465
ptr2
     loader         1422  1456  1456  1458  1459  1464  1465
ptrmp
     srm_types       823   875
ptrrareyoualive
     srm_types       825   877
ptrrcat
     srm_types       827   879
ptrrcatpass
     srm_types       829   881
ptrrchangeprotect
     srm_types       831   883
ptrrchangevolume
     srm_types       833   885
ptrrclose
     srm_types       835   887
ptrrcopy
     srm_types       837   889
ptrrcreatefile
     srm_types       839   891
ptrrcreatelink
     srm_types       841   893
ptrrexchange
     srm_types       843   895
ptrrfileinfo
     srm_types       845   897
ptrrgangclean
     srm_types       847   899
ptrrhead
     srm_types       872   923
ptrrinit
     srm_types       849   901
ptrrlock
     srm_types       851   903
ptrropen
     srm_types       853   905
ptrrpos
     srm_types       855   907
ptrrpurge
     srm_types       857   909
ptrrread
     srm_types       859   911
ptrrsetdate
     srm_types       861   913
ptrrseteof
     srm_types       863   915
ptrrunlock
     srm_types       865   917
ptrrvol
     srm_types       867   919
ptrrwrite
     srm_types       869   921
ptrsareyoualive
     srm_types       824   876
ptrscat
     srm_types       826   878
ptrscatpass
     srm_types       828   880
ptrschangeprotect
     srm_types       830   882
ptrschangevolume
     srm_types       832   884
ptrsclose
     srm_types       834   886
ptrscopy
     srm_types       836   888
ptrscreatefile
     srm_types       838   890
ptrscreatelink
     srm_types       840   892
ptrsexchange
     srm_types       842   894
ptrsfileinfo
     srm_types       844   896
ptrsgangclean
     srm_types       846   898
ptrshead
     srm_types       871   922
ptrsinit
     srm_types       848   900
ptrslock
     srm_types       850   902
ptrsopen
     srm_types       852   904
ptrspos
     srm_types       854   906
ptrspurge
```

```
                    253
sizechk
     lifdam          99   102
sizeof
     loader        1146  1235  1320  1392  1433  1441  1458  1464  1523  1530  1602  1604
     initload      1675
     ldr            949   951
     lifdam         244   336   509  1124
     srm_drv       1373  1418
     srmdam         100  1398
     librarian      166   168   172   195   322  1255  1360  1374  1584  1589  1596  1600  1611  1616  1618  1626  1627  1632
                   1664  1675  1681  1681  1682  1696  1707  1721  1722  1726  1727  1729  1778  1819  1829  1905  1911  1921
                   1922  1958  1959  1977  2003  2005  2012  2021  2028  2031  2104  2114  2251  2256  2286  2388  2409  2452
                   2455  2463  2475  2480  2549  2597  2626  2627  2682  2717  2722  2732  2739  2746  2752  2759  2784  2822
                   2829  2848  2851  2864  2865  2870  2871  2876  2883  2888  2890  2900  2900  2964  3031  3143  3143  3226
                   3290
     hminit         168
     qminit         557   584   609   636   668   671   714   718   755   780
     miui           482   486
     tapebkup        56    83   116   139   169   213
     heapt          126
     amigo          235   292   312   335   414
     cs80           314   333   350   367   404   432   454   470   488   506   545
size_from_are_you_alive
     packets         89
size_from_cat
     packets         90
size_from_catprotect
     packets         91
size_from_changeprotect
     packets         92
size_from_change_vol_label
     packets         93
size_from_close
     packets         94
size_from_copy
     packets         95
size_from_create
     packets         96
size_from_createlink
     packets         97
size_from_flock
     packets         98
size_from_funlock
     packets         99
size_from_gang_cleanup
     packets        100
size_from_gang_error
     packets        101
     srm_drv       1428  1431  1438
size_from_info
     packets        102
size_from_init_vol
     packets        103
size_from_open
     packets        104
size_from_position
     packets        105
size_from_purge
     packets        106
size_from_read
     packets        107
     srm_drv       1438
size_from_set_date
     packets        109
size_from_set_eof
     packets        108
size_from_volstatus
     packets        110
size_from_write
     packets        111
size_from_xchg_data
     packets        112
size_from_xchg_open
     packets        113
size_to_are_you_alive
     packets         57
     srm_drv       1529
size_to_cat
     packets         58
     srm_drv       1556
size_to_catprotect
     packets         59
     srm_drv       1594
size_to_changeprotect
     packets         60
     srm_drv       1633
size_to_change_vol_label
     packets         62
     srm_drv       1661
size_to_close
     packets         63
     srm_drv       1682
size_to_copy
     packets         64
     srm_drv       1707
size_to_create
     packets         65
     srm_drv       1745
size_to_createlink
     packets         67
     srm_drv       1794
size_to_flock
     packets         68
     srm_drv       1884
size_to_funlock
     packets         69
     srm_drv       2046
size_to_gang_cleanup
     packets         70
     srm_drv       1865
size_to_info
     packets         71
     srm_drv       1846
```

```
     hminit          287   289
text
     globals          33
     init           1090  1091
     convert          35    36    40    87
     librarian        37    44   102   217  2406
     xminit         1230
     tapebkup        764
textbuf
     librarian      1451  1464  1473  1475  1479  1483
textbufblocks
     librarian      2226  2583  2586  2589  2589  2590  2590  2592
textbuffer
     loader         1500  1515  1516  1563
     librarian      2204  2247  2249  2274  2276  2277  2302  2307  2307  2308  2320  2321  2525  2592  2618  2669
textbuftop
     librarian      2205  2316  2593  2617  2668
textdescptr
     loader          873   900
textdescriptor
     loader          873   988  1530
     librarian      1374  1958  1959  2003  2028  2031  2251  2286  2784  2871
textfield
     crt              68   206   217   437
textfile
     globals          49
     librarian      2996
textinblock
     librarian      2227  2328  2329  2329  2703
textindex
     librarian      2206  2247  2274  2276  2304  2304  2308  2314  2316  2320  2321  2328  2331  2331  2617  2618  2662  2665
                    2666  2668  2669  2670  2671  2671  2777
textinsize
     librarian      2228  2317  2317  2330  2330  2701  2777
textoutblock
     librarian      2229  2243  2277  2302  2303  2303
textoutsize
     librarian      2230  2247  2271  2674  2674  2704  2704
textptr
     globals          33   257
textrecctr
     librarian      1356  1364  1365  1367  1367  1369
textrecords
     loader          962  1527
     librarian       378  1364  1369  2025  2061  2692  2835  2892  2892
textrecs
     librarian      1939  2025  2026  2035  2035
textsize
     loader          990  1532  1539
     librarian      1371  1378  1379  1380  1387  1953  1960  1966  1966  1966  2004  2029  2029  2029  2030  2030  2244  2271
                    2291  2694  2701  2704  2884  2886  2887  2888  2890
textstart
     loader          989  1539
     librarian      1370  1379  2243  2703  2887
textstep
     librarian       315   322   324   325   349   350   350   353   355  1355  1360  1361  1362  1362  1363  1363  1363  1365
                    1374  1374  1376  1377
texttype
     srmdam            70
tfname
     bootdam          651   662   662   665
tfr_data_sec
     amigo            762   766
tfr_end
     srm_drv         1299  1305  1348
thedate
     clock              41    43    87   103   147   151
thetime
     clock              42    44    59    78    82   118   122   130   139   140   161   165   174   186   187   197   203   207
thirdparm
     librarian      3190  3192  3211  3218
this
     heapt            133   147   148   153   154   157
thislink
     heapt            133   148   149   150   153   154   156
this_device
     amigo            205   207   208   215
this_entry_ptr
     qminit           963  1032  1033  1046  1077
tid
     globals           42   102
     loader          1018
     misc             181
     ldr              926
     srmdam          1586
tidleng
     globals           19    42
     bootdam          697
     ldr              929
     lifdam           216  1051
     srmdam          1623
time
     globals          231
     lifdam           529   536   537   538
     srm_drv         1113  1114  1115  1234  1236  1240  1242  1246  1248  1253  1257  1259
     srmdam           335   347   348   349   350  1420
     librarian        218   230   231
     miui              53    62    78    80    81    83    84
     iolib            144   228   230   235   235   244
time1
     cs80             285   494   504
time2
     cs80             285   494   505
timeout
     srm_drv         1259
     kernel           393   407   421  1130  1188
     iolib            246   415   441   492   518   578   628
     printer          114   244   252
     dischpib         265   323
timeouts_per_beep
     printer           56   216
timeout_blanked
     printer           76   196   206   210   210
timeout_counter
```

```
      lifdam          208
      hminit          424   430
      tapebkup        432   626
      f9885           138   280
      amigo           605   666   902
      cs80            629  1010  1107
znoblock
      globals         191
      mini            351   357   360   361   362   363
      bootdam         758
      misc            334
      miui            119
      tapebkup        425   842
      f9885           228   235
      amigo           824   870   876   922
      cs80            617
znodevice
      globals         187
      mini            370   483   490   493
      bootdam         597
      misc            318
      initunits       804   807
      srm_drv        1151  1459
      srmdam         1599
      srmam           151   159   188
      qminit          859
      tapebkup        249   280   303   346   509   703   735
      printer         105   140   308
      f9885            89   217   255   262
      dischpib        218   237
      amigo           172   470   478   483   516   519   529   533   535   567   879   891
      cs80            601   843   847   974  1053  1073  1241
znomedium
      globals         191
      mini            345   346   347   348   350   367
      bootdam         598
      misc            336
      hminit          431
      f9885           220
      amigo           892
znosuchblk
      globals         187
      misc            317
      tapebkup        420
      amigo           921
      cs80            604
znotready
      globals         191
      bootdam         599
      misc            335
      srm_drv        1180
      tapebkup        422
      amigo           890
      cs80            612
zprotected
      globals         187
      mini            358   359
      misc            320
      hminit          432
      f9885           224
      amigo           881   896
      cs80            614
zstrangei
      globals         188
      mini            365   366
      misc            321
ztimeout
      globals         184
      misc            306
      srm_drv        1381  1464
      printer         132   224   334
      dischpib        221
zuninitialized
      globals         191
      misc            333
      miui            120
      tapebkup        843
      amigo           669   898
      cs80            608
z_type
      cs80            224   282   438   445

      4354 identifiers,     40336 occurrences.
```

Assembly Source Listings

This volume of the System Internals Documentation contains the source text for the following Assembly modules:

- ALPHALIST
- ASM
- BOOTDEFS
- COMASM
- DC
- DISCINT
- DRVASM
- EVALGVR
- FASTMOVE
- FMINIT
- FORMAT
- GPIO
- GPIODVR
- MATCH
- MATCHSTR
- MODIV
- NEWWORDS
- POWERUP
- RANDOM
- ROMCALL
- RS
- RSTRINT
- SCAN
- SETSTUFF
- STRG1
- STRG2
- STRINT
- SYSDEF

ALPHALIST

Purpose

ALPHALIST alphabetizes a list of symbols.

Usage

ALPHALIST is used by the linking loader.

```
 PAGE   1 [2.0] 11/4/82 13:14:18  ASSEMBLY OF ALPHALIST.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                           *       procedure alphalist
    2                           *               var symtable: array[0..65535] of char; {symbol table}
    3                           *               var list:     array[0..N-1]   of 0..65535;
    4                           *                   N:        0..65535;
    5                           *
    6                           *       The N elements of the list are offsets into the symbol table,
    7                           *       where strings are found.  The object is to rearrange the list
    8                           *       so that the strings are alphabetized.
    9
   10    00000000                       rorg    0
   11
   12                                   def     alphalist
   13
   14              0000 0000 symtable equ     a0              base of symbol table containing strings
   15              0000 0001 I        equ     a1              index into list
   16              0000 0002 J        equ     a2              index into list
   17              0000 0003 Istr     equ     a3              address of string
   18              0000 0004 Jstr     equ     a4              address of string
   19
   20              0000 0000 Ioffset equ     d0              offset into symbol table
   21              0000 0001 Joffset equ     d1              offset into symbol table
   22              0000 0002 Icount  equ     d2              number of elements processed
   23              0000 0003 Jcount  equ     d3              inner loop counter
   24              0000 0004 N       equ     d4              size of list
   25              0000 0005 len     equ     d5              length of string
   26              0000 0006 comp    equ     d6              condition codes
   27
   28              0000 0000 alphalist equ   *
   29
   30    00000000 245F               movea.l (sp)+,J                 return address
   31    00000002 381F               move.w  (sp)+,N                 size of list
 . 32    00000004 225F               movea.l (sp)+,I                 address of list
   33    00000006 205F               movea.l (sp)+,symtable          address of symbol table
   34    00000008 2F0A               move.l  J,-(sp)                 replace return address
   35
   36    0000000A 5544               subq.w  #2,N                    list must have least two elements
   37    0000000C 654C               bcs.s   done
   38    0000000E 5489               addq.l  #2,I                    start with second list element
   39    00000010 4280               clr.l   Ioffset                 sign extend to long
   40    00000012 4281               clr.l   Joffset                 sign extend to long
   41    00000014 4242               clr.w   Icount                  initialize tally of elements processed
   42    00000016 2449       I_loop  movea.l I,J                     for J := I
   43    00000018 3602               move.w  Icount,Jcount                   downto 0
   44    0000001A 3019               move.w  (I)+,Ioffset            ioffset := list[I]
   45    0000001C 5242               addq.w  #1,Icount                       I := I + 1
   46    0000001E 3222       J_loop  move.w  -(J),Joffset            J := J - 1; joffset := list[J]
   47
   48    00000020 49F0 1800          lea     0(symtable,Joffset.l),Jstr      standard string comparison:
   49    00000024 47F0 0800          lea     0(symtable,Ioffset.l),Istr
   50    00000028 4245               clr.w   len                     sign extend to word
   51    0000002A 1A1B               move.b  (Istr)+,len
   52    0000002C BA1C               cmp.b   (Jstr)+,len             compare lengths
   53    0000002E 40C6               move    sr,comp                 save comparison till later
   54    00000030 6304               bls.s   str_comp
   55    00000032 1A2C FFFF          move.b  -1(Jstr),len            get minimum of two string lengths
   56    00000036 5305       str_comp subq.b #1,len                  loop if at least one character
   57    00000038 6508               bcs.s   cmp_end
   58    0000003A B70C       cmp_lp  cmpm.b  (Jstr)+,(Istr)+         compare string bodies

 PAGE   2 [2.0] 11/4/82 13:14:18  ASSEMBLY OF ALPHALIST.TEXT

   59    0000003C 56CD FFFC          dbne    len,cmp_lp              loop till not equal or end of string
   60    00000040 6602               bne.s   cmp                     if string bodies are equal,
   61    00000042 44C6       cmp_end move    comp,ccr                        then compare lengths
   62    00000044 640C       cmp     bcc.s   found                   if string(I) >= string(J), J loop done
   63
   64    00000046 3541 0002          move.w  Joffset,2(J)            list[J+1] := joffset
   65    0000004A 51CB FFD2          dbra    Jcount,J_loop           loop until J = 0
   66    0000004E 3480               move.w  Ioffset,(J)             if J = 0 then list[0] := ioffset
   67    00000050 6004               bra.s   next_I
   68
   69    00000052 3540 0002  found   move.w  Ioffset,2(J)            list[J+1] := ioffset
   70    00000056 51CC FFBE  next_I  dbra    N,I_loop                loop until N = 0
   71
   72    0000005A 4E75       done    rts                             wasn't that easy?
   73
   74                                end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

*** NO EXTERNAL SYMBOLS ***

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ALPHALIST	REL	28		00000000
CCR	STREG	0		00000005
CMP	REL	62		00000044
CMP_END	REL	61		00000042
CMP_LP	REL	58		0000003A
COMP	DREG	26		00000006
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DONE	REL	72		0000005A
FOUND	REL	69		00000052
I	AREG	15		00000001
ICOUNT	DREG	22		00000002
IOFFSET	DREG	20		00000000
ISTR	AREG	17		00000003
I_LOOP	REL	42		00000016
J	AREG	16		00000002
JCOUNT	DREG	23		00000003
JOFFSET	DREG	21		00000001
JSTR	AREG	18		00000004
J_LOOP	REL	46		0000001E
LEN	DREG	25		00000005
N	DREG	24		00000004
NEXT_I	REL	70		00000056
SP	AREG	0		00000007
SR	STREG	0		00000006
STR_COMP	REL	56		00000036
SYMTABLE	AREG	14		00000000
USP	STREG	0		00000007

ASM

Purpose

ASM provides the Pascal declarations for a number of routines which are in assembly. ASM also contains initialization and utilities used during the boot-up process.

Usage

ASM is part of SYSTEM_P.

```
PASS 1 COMPLETE. ERRORS: 0
    1                                   MNAME   ASM
    2                                   SPRINT
    3
    4                                   DEF     ASM_ASM,ASM_POWERUP,ASM_ERRMSG,G_DOLLAR,STACKFUDGE
    5                                   DEF     ASM_USERPROGRAM,ASM_CI_SWITCH,ASM_FINDROMS,ASM_CPYMSG
    6
    7
    8                           SRC MODULE ASM;
    9                           SRC IMPORT SYSGLOBALS;
   10                           SRC EXPORT
   11                           SRC TYPE
   12                           SRC STRINGMAX = STRING[255];
   13                           SRC   PROCEDURE MOVELEFT (ANYVAR S,D: INTEGER; Z:INTEGER);
   14                           SRC   PROCEDURE MOVERIGHT(ANYVAR S,D: INTEGER; Z:INTEGER);
   15                           SRC   PROCEDURE FASTMOVE (       S,D: ANYPTR;  Z:INTEGER);
   16                           SRC   PROCEDURE NEWBYTES(VAR P: ANYPTR;        Z:INTEGER);
   17                           SRC   PROCEDURE POWERUP;
   18                           SRC   PROCEDURE ERRMSG;
   19                           SRC   PROCEDURE FINDROMS;
   20                           SRC   PROCEDURE F_PWR_ON;
   21                           SRC   PROCEDURE FLPYREAD(SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   22                           SRC   PROCEDURE FLPY_WRT(SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   23                           SRC   PROCEDURE FLPYMREAD(SECTOR_COUNT, SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   24                           SRC   PROCEDURE FLPYMWRITE(SECTOR_COUNT, SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   25                           SRC   PROCEDURE FLPYINIT(PTR: ANYPTR; I: SHORTINT);
   26                           SRC   PROCEDURE SETINTLEVEL(LEVEL: INTEGER);
   27                           SRC   FUNCTION  INTLEVEL:  INTEGER;
   28                           SRC   PROCEDURE NEWWORDS(VAR P: ANYPTR; WORDSIZE: INTEGER);
   29                           SRC   PROCEDURE USERPROGRAM (EXECLOC,INITSP: INTEGER);
   30                           SRC   PROCEDURE SAPPEND(VAR DEST: STRING;SRC:STRINGMAX);
   31                           SRC   FUNCTION IAND(A,B: INTEGER): INTEGER;
   32                           SRC   FUNCTION IOR (A,B: INTEGER): INTEGER;
   33                           SRC   PROCEDURE CI_SWITCH;
   34                           SRC   PROCEDURE INITVECTS;
   35                           SRC   PROCEDURE CPYMSG(MSG: STRING255);
   36                           SRC END;
   37
   38
   39                                   REFR    FS_FWRITESTRINT,INITLOAD_INITLOAD
   40                                   REFA    SYSGLOBALS,LOADER
   41             FFFF FFFE ESCAPECODE  EQU SYSGLOBALS-2
   42             FFFF FFF6 RECOVERBLOCK EQU SYSGLOBALS-10
   43             FFFF FFF2 HEAPPOINTER EQU SYSGLOBALS-14
   44             FFFF FFEE HEAPBASE    EQU SYSGLOBALS-18
   45             FFFF FFEA IORESULT    EQU SYSGLOBALS-22
   46             FFFF FFBA SYSDEFS     EQU LOADER-70              LINKED LIST OF PERMANT PROGRAMS
   47
   48             0000 1388 SUPSTACKSPACE EQU 5000        SPACE FOR SUPERVISOR STACK
   49             0000 01F4 STACKFUDGE  EQU 500           SLOP SPACE FOR STACK OVERFLOW CHECK
   50
   51             FFFF FB2E G_DOLLAR    EQU $FFFFFB2E     MAGIC NUMBERS, CONSULT 'POWERUP' LISTING
   52             FFFF FB22 INITSTACK   EQU $FFFFFB22
   53             FFFF FB26 INITPC      EQU $FFFFFB26
   54             FFFF FB2A INITRECOVER EQU $FFFFFB2A
   55             FFFF FB48 INITSR      EQU $FFFFFB48
   56
   57             FFFF FDCE LOWMEM      EQU $FFFFFDCE     BOOT ROM DEFINED ADDRESSES
   58             FFFF FB00 HIGHMEM     EQU $FFFFFB00     LEAVES ROOM FOR VECTORS, MONITOR STUFF, ETC.
```



```
   59             FFFF FF52 TRAP11      EQU $FFFFFF52
   60             FFFF FF58 TRAP10      EQU $FFFFFF58
   61             FFFF FF94 TRAP0       EQU $FFFFFF94
   62
   63
   64             0000 0150 CRTMSG      EQU $150
   65                       *CRTCLEAR     EQU $148       CAN'T USE DUE TO BOOT ROM BUG
   66             FFFF FED2 SYSFLAG     EQU -302
   67
   68             0000 4EF9 JMP         EQU $4EF9
   69             0000 0004 RETURN      EQU A4
   70
   71
   72
   73
   74             0000 0000 LOWCODE EQU *                          LOWEST CODE LOCATION
   75             0010 0000 HIGHCODE EQU *+$100000                 1 MEGABYTE MAX ROM AREA
   76
   77    00000000 F0                    DC.B    $F0,$FF,'P',1      ROM HEADER
   78    00000004 0000 0000             DC.L    0,INITLOAD_INITLOAD-*  CHECKSUM, EXECUTION ADDRESS
   79    0000000C 00                    DC.B    0,0,0,0,0,0        IDENTIFICATION JUNK
   80
   81
   82             0000 0012 ASM_POWERUP EQU *
   83    00000012 285F                  MOVEA.L (SP)+,RETURN
   84    00000014 4FFA FFEA             LEA     LOWCODE,SP         SOFT SYSTEM STACK = JUST BELOW CODE
   85    00000018 BFF8 FDCE             CMPA.L  LOWMEM,SP          TEST FOR HARD OR SOFT SYSTEM
   86    0000001C 6204                  BHI.S   SOFT               SOFT IF SP > LOWMEM^
   87    0000001E 4FF8 FB00             LEA     HIGHMEM,SP         HARD SYSTEM STACK = HIGH MEMEMORY
   88             0000 0022 SOFT       EQU *
   89
   90    00000022 4BEF EC78             LEA     -SUPSTACKSPACE(SP),A5  ALLOCATE OPERATING SYSTEM STACK
   91    00000026 4BED 8000             LEA     -32768(A5),A5      SYSTEM GLOBALS START AT 32766(A5)
   92    0000002A 21CD FB2E             MOVE.L  A5,G_DOLLAR        SAVE FOR ISR's
   93
   94    0000002E 2B78 FDCE             MOVE.L  LOWMEM,HEAPPOINTER(A5)  INITIALIZE HEAP
                  FFF2
   95
   96    00000034 46FC 2000             MOVE    #$2000,SR          ENABLE INTERRUPTS
   97
   98    00000038 41F8 FF52             LEA     TRAP11,A0          SUPERVISOR CALL TRAP
   99    0000003C 30FC 4EF9             MOVE    #JMP,(A0)+
  100    00000040 43FA 0026             LEA     SUPERCALL,A1
  101    00000044 20C9                  MOVE.L  A1,(A0)+
  102
  103                      *            LEA     TRAP10,A0          ESCAPE(N) TRAP
  104    00000046 30FC 4EF9             MOVE    #JMP,(A0)+
  105    0000004A 43FA 0016             LEA     ESCN,A1
  106    0000004E 20C9                  MOVE.L  A1,(A0)+
  107
  108    00000050 41F8 FF94             LEA     TRAP0,A0           COMPILED LINE TRAP
  109    00000054 30FC 4EF9             MOVE    #JMP,(A0)+
  110    00000058 43FA 001C             LEA     P_BREAK,A1
  111    0000005C 20C9                  MOVE.L  A1,(A0)+
  112
  113    0000005E 4ED4                  JMP     (RETURN)           ALL DONE
  114
```

```
00000060 4E75      ASM_ASM RTS

00000062 2E6D FFF6 ESCN    MOVEA.L RECOVERBLOCK(A5),SP
00000066 4E75              RTS

           0000 0068 SUPERCALL EQU *                  TRAP #11, GETS INTO SUPERVISOR MODE, SAVES SR
00000068 2F2F 0002         MOVE.L  2(SP),-(SP)     COPY RETURN ADDRESS
0000006C 3F6F 0004         MOVE    4(SP),8(SP)     MOVE STATUS REGISTER UP
           0008
00000072 2E9F              MOVE.L  (SP)+,(SP)      MOVE RETURN ADDRESS UP
00000074 4E75              RTS                     RETURN TO CALLER IN SUPERVISOR MODE

00000076 54AF 0002 P_BREAK ADDQ.L  #2,2(SP)
0000007A 4E73              RTE

           0000 007C ASM_USERPROGRAM EQU     *        PROC USERPROGRAM(ENTRYPOINT,INITSTACK:INTEGER)
0000007C 265F              MOVEA.L (SP)+,A3        RETURN ADDRESS
0000007E 205F              MOVEA.L (SP)+,A0        INITIAL SP
00000080 225F              MOVEA.L (SP)+,A1        INITIAL PC
00000082 2F0B              MOVE.L  A3,-(SP)        RESTORE RETURN

00000084 45E8 FE0C         LEA     -STACKFUDGE(A0),A2
00000088 B5ED FFF2         CMPA.L  HEAPPOINTER(A5),A2      COMPARE
0000008C 6202              BHI.S   *+4                     TEST
0000008E 4E42              TRAP    #2                      STACK OVERFLOW

00000090 2F2D FFF6         MOVE.L  RECOVERBLOCK(A5),-(SP)  TRY
00000094 2F0E              MOVE.L  A6,-(SP)                SAVE A6
00000096 487A 002E         PEA     USER_RECOVER            RECOVER CODE
0000009A 2B4F FFF6         MOVE.L  SP,RECOVERBLOCK(A5)

0000009E 21CF FB2A         MOVE.L  SP,INITRECOVER  SAVE FOR DEBUGGER
000000A2 21C9 FB26         MOVE.L  A1,INITPC       SAVE FOR DEBUGGER

000000A6 4DD5              LEA     (A5),A6         FRAME POINTER
000000A8 4E60              MOVE    A0,USP          SET UP STACK
000000AA 46FC 0000         MOVE    #$0000,SR       GO INTO USER MODE

000000AE 45EF FFF8         LEA     -8(SP),A2       SPACE FOR RETURN ADDRESS, DYNAMIC LINK
000000B2 21CA FB22         MOVE.L  A2,INITSTACK    SAVE NON-LOCAL GOTO
000000B6 40F8 FB48         MOVE    SR,INITSR       SAVE FOR STOP KEY
000000BA 42AD FFEE         CLR.L   HEAPBASE(A5)    FOR MEMORY MANAGER

000000BE 4E91              JSR     (A1)            CALL USER PROGRAM

000000C0 426D FFFE         CLR.W   ESCAPECODE(A5)  NORMAL EXIT
000000C4 4E4A              TRAP    #10             ESCAPE(0), DONE TO CLOSE FILES
           0000 00C6 USER_RECOVER    EQU *
000000C6 2C5F              MOVEA.L (SP)+,A6        RESTORE A6
000000C8 2B5F FFF6         MOVE.L  (SP)+,RECOVERBLOCK(A5)  RESTORE RECOVER BLOCK
000000CC 204F              MOVEA.L SP,A0           SAVE STACK POINTER
000000CE 4E4B              TRAP    #11             GET INTO SUPERVISOR MODE
000000D0 2E48              MOVEA.L A0,SP           RESTORE STACK POINTER
000000D2 4E75              RTS

           0000 00D4 ASM_CI_SWITCH EQU *
000000D4 205F              MOVEA.L (SP)+,A0        RETURN ADDRESS
```



```
000000D6 4E4B              TRAP    #11             GET ONTO SUPERVISOR STACK
000000D8 544F              ADDQ    #2,SP           DISCARD SR
000000DA 4ED0              JMP     (A0)            DONE

           0000 4000 K16      EQU     $4000           16 K INCREMENTS
           0000 0012 MDLINK   EQU     18              OFFSET OF MODULE DESCRIPTOR
           0000 0028 MDSIZE   EQU     40              SIZE OF DESCRIPTOR

           0000 00DC ASM_FINDROMS EQU *
000000DC 41FA FF22         LEA     LOWCODE,A0              BEGINNING OF SEARCH AREA
000000E0 226D FFF2         MOVEA.L HEAPPOINTER(A5),A1      HEAP START
000000E4 202D FFBA         MOVE.L  SYSDEFS(A5),D0          LINKED LIST HEAD

000000E8 0C90 F0FF SEARCH  CMPI.L  #$F0FF5080,(A0)         PASCAL OPTION?
           5080
000000EE 6624              BNE.S   NOROM                   NO MATCH
000000F0 45E8 0012         LEA     MDLINK(A0),A2           LOCATE DESCRIPTOR

000000F4 4CEA 18FE COPY    MOVEM.L 4(A2),D1-D7/A3-A4       REMAINDER OF MDB
           0004
000000FA 48D1 18FF         MOVEM.L D0-D7/A3-A4,(A1)        COPY ONTO HEAP
000000FE 2009              MOVE.L  A1,D0                   INSERT LINK INTO LIST
00000100 D3FC 0000         ADDA.L  #MDSIZE,A1              ALLOCATE FROM HEAP
           0028
00000106 2B49 FFF2         MOVE.L  A1,HEAPPOINTER(A5)      RESTORE HEAP
0000010A 2B40 FFBA         MOVE.L  D0,SYSDEFS(A5)          RESTORE LIST

0000010E 2452              MOVEA.L (A2),A2         MOVE DOWN CHAIN
00000110 220A              MOVE.L  A2,D1           TEST FOR NIL
00000112 66E0              BNE.S   COPY            NOT END OF CHAIN
00000114 D0FC 4000 NOROM   ADDA.W  #K16,A0         NEXT ROM BOUNDARY
00000118 B1FC 0010         CMPA.L  #HIGHCODE,A0    END OF ROM AREA?
           0000
0000011E 66C8              BNE.S   SEARCH          NOPE
00000120 4E75              RTS


                   ****************  CRT CLEAR ROUTINE ****************************************
           0000 0122 CRTCLEAR        EQU *
00000122 4E56 FFAE         LINK    A6,#-82
00000126 41D7              LEA     (SP),A0
00000128 704F              MOVEQ   #79,D0
0000012A 0838 0000         BTST    #0,SYSFLAG      TEST FOR 50 CHAR SCREEN
           FED2
00000130 6702              BEQ.S   LC1
00000132 7031              MOVEQ   #49,D0
00000134 10FC 0020 LC1     MOVE.B  #' ',(A0)+
00000138 51C8 FFFA         DBRA    D0,LC1
0000013C 4210              CLR.B   (A0)
0000013E 422E FFFF         CLR.B   -1(A6)
00000142 41D7      LC2     LEA     (SP),A0
00000144 4240              CLR     D0
00000146 102E FFFF         MOVE.B  -1(A6),D0
0000014A 4EB8 0150         JSR     CRTMSG
0000014E 522E FFFF         ADDQ.B  #1,-1(A6)
00000152 0C2E 0018         CMPI.B  #24,-1(A6)
           FFFF
```

```
PAGE   5 [2.0] 11/4/82 13:13:00  ASSEMBLY OF ASM.TEXT

 222    00000158 66E8              BNE.S   LC2
 223    0000015A 4E5E              UNLK    A6
 224    0000015C 4E75              RTS
 225
 226                       ****************  INTRODUCTORY MESSAGE ROUTINE  ****************************
 227
 228              0000 015E ASM_CPYMSG      EQU *
 229    0000015E 4EBA FFC2          JSR     CRTCLEAR
 230    00000162 7000              MOVEQ   #0,D0
 231    00000164 41FA 0088         LEA     M,A0                    COPYRIGHT NOTICE
 232    00000168 4EB8 0150         JSR     CRTMSG
 233    0000016C 7002              MOVEQ   #2,D0
 234    0000016E 225F              MOVEA.L (SP)+,A1
 235    00000170 205F              MOVEA.L (SP)+,A0
 236    00000172 4241              CLR     D1
 237    00000174 1218              MOVE.B  (A0)+,D1
 238    00000176 4230 1000         CLR.B   0(A0,D1)
 239    0000017A 2F09              MOVE.L  A1,-(SP)
 240    0000017C 4EB8 0150         JSR     CRTMSG
 241    00000180 4E75              RTS
 242
 243                       ****************  PRIMITIVE ERROR MESSAGE ROUTINE  *************************
 244
 245              FFFF FFAA STR     EQU -86
 246              FFFF FFFC I       EQU -4
 247
 248    00000182 4281      INT     CLR.L   D1
 249    00000184 122E FFAA         MOVE.B  STR(A6),D1
 250    00000188 5281              ADDQ.L  #1,D1
 251    0000018A 2D41 FFFC         MOVE.L  D1,I(A6)
 252    0000018E 1F3C 0050         MOVE.B  #80,-(SP)
 253    00000192 486E FFAA         PEA     STR(A6)
 254    00000196 486E FFFC         PEA     I(A6)
 255    0000019A 2F00              MOVE.L  D0,-(SP)
 256    0000019C 3F3C FFFF         MOVE.W  #-1,-(SP)
 257    000001A0 4EBA FE5E         JSR     FS_FWRITESTRINT
 258    000001A4 4E75              RTS
 259
 260              0000 01A6 ASM_ERRMSG  EQU *
 261    000001A6 4E4B              TRAP    #11
 262    000001A8 4E56 FFAA         LINK    A6,#STR
 263    000001AC 4EBA FF74         JSR     CRTCLEAR
 264    000001B0 4CFA 003E         MOVEM.L M1,D1-D5
                 0062
 265    000001B6 48EE 003E         MOVEM.L D1-D5,STR(A6)
                 FFAA
 266    000001BC 202D FFEA         MOVE.L  IORESULT(A5),D0
 267    000001C0 61C0              BSR.S   INT
 268    000001C2 302D FFFE         MOVE.W  ESCAPECODE(A5),D0
 269    000001C6 48C0              EXT.L   D0
 270    000001C8 61B8              BSR.S   INT
 271    000001CA 4280              CLR.L   D0
 272    000001CC 41EE FFAA         LEA     STR(A6),A0
 273    000001D0 4241              CLR     D1
 274    000001D2 1210              MOVE.B  (A0),D1
 275    000001D4 4230 1001         CLR.B   1(A0,D1)
 276    000001D8 5248              ADDQ    #1,A0
```

```
PAGE   6 [2.0] 11/4/82 13:13:00  ASSEMBLY OF ASM.TEXT

 277    000001DA 4EB8 0150         JSR     CRTMSG
 278
 279    000001DE 701C              MOVEQ   #28,D0
 280    000001E0 51C9 FFFE L       DBRA    D1,L
 281    000001E4 51C8 FFFA         DBRA    D0,L
 282
 283    000001E8 4E5E              UNLK    A6
 284    000001EA 46DF              MOVE    (SP)+,SR
 285    000001EC 4E75              RTS
 286
 287    000001EE 43        M       DC.B    'Copyright 1982 Hewlett-Packard Company.',0
 288    00000216 12        M1      DC.B    18,'IORESULT, ERROR = '
 289
 290
 291                               NOSYMS
 292                               END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

BOOTDEFS

Purpose

BOOTDEFS defines several entry points and table addresses within the Boot ROM.

```
 PAGE   1 [2.0] 11/4/82 13:14:37  ASSEMBLY OF BOOTDEFS.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1    00000000                           rorg 0
    2
    3                                       nosyms
    4                         * This file contains hardware dependent addresses for the
    5                         *
    6                         *                   manufacturing unit
    7                         *
    8                         *                       Chipmunks
    9
   10                         *
   11                         *  The following are the addresses of the coefficients used in the
   12                         *  evaluation of Basic functions.
   13                         *
   14              0000 3C26 cff_loga     equ   $3c26             LOG coefficients
   15              0000 3C3E cff_logb     equ   $3c3e
   16
   17              0000 3C56 cff_expp     equ   $3c56             EXP coefficients
   18              0000 3C6E cff_expq     equ   $3c6e
   19
   20              0000 3C8E cff_sin      equ   $3c8e             SIN/COS coefficients
   21
   22              0000 3CCE cff_tanp     equ   $3cce             TAN coefficients
   23              0000 3CEE cff_tanq     equ   $3cee
   24
   25              0000 3D16 cff_asnp     equ   $3d16             ASN/ACS coefficients
   26              0000 3D3E cff_asnq     equ   $3d3e
   27
   28              0000 3D66 cff_atnp     equ   $3d66             ATN coefficients
   29              0000 3D86 cff_atnq     equ   $3d86
   30
   31              0000 3DA6 cff_powp     equ   $3da6             x^y coefficients
   32              0000 3DC6 cff_powq     equ   $3dc6
   33
   34                         *
   35                         *  The following are address of tables used in the BCD <-> real
   36                         *  conversions and in the evaluation of x^y.
   37                         *
   38              0000 3658 tb_pwt       equ   $3658             BCD <-> real tables
   39              0000 3698 tb_pwt8      equ   $3698
   40              0000 36B8 tb_pwt4      equ   $36b8
   41              0000 36D8 tb_pwtt      equ   $36d8
   42              0000 3AE0 tb_auxpt     equ   $3ae0
   43              0000 3B28 tb_bcd       equ   $3b28
   44              0000 3BC2 tb_bin       equ   $3bc2
   45              0000 3DFE tb_a1        equ   $3dfe             x^y tables
   46              0000 3E8E tb_a2        equ   $3e8e
   47
   48                         *
   49                         * The following are compiler support routines
   50                         * in the boot rom.
   51                         *
   52                                 def     asm_assign,asm_difference
   53                                 def     asm_equal,asm_in,asm_inclusion
   54                                 def     asm_intersect,asm_mpy,asm_nequal
   55                                 def     asm_union
   56                                 def     asm_rmovel,asm_rmover
   57                                 def     asm_pos
   58
```

```
 PAGE   2 [2.0] 11/4/82 13:14:37  ASSEMBLY OF BOOTDEFS.TEXT

   59              0000 3372 asm_assign           equ     $3372
   60              0000 3488 asm_difference       equ     $3488
   61              0000 330A asm_equal            equ     $330a
   62              0000 34DA asm_in               equ     $34da
   63              0000 33F4 asm_inclusion        equ     $33f4
   64              0000 344A asm_intersect        equ     $344a
   65              0000 31A6 asm_mpy              equ     $31a6
   66              0000 3300 asm_nequal           equ     $3300
   67              0000 3398 asm_union            equ     $3398
   68
   69              0000 3108 asm_rmovel           equ     $3108
   70              0000 315A asm_rmover           equ     $315a
   71              0000 361E asm_pos              equ     $361e
   72
   73                                    end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

COMASM

Purpose

COMASM contains common I/O driver assembly language support routines and binary operations.

Usage

The buffer, interrupt and DMA facilities are used by HPIB, GPIO and RS232 low-level drivers, The binary operations are exported for common usage (and are used in the I/O library).

Requirements

I/O library kernel (IODECLARATIONS, etc.)

```
 PAGE   1 [2.0] 11/4/82 16:16:33  ASSEMBLY OF COMASM.TEXT
PASS 1 COMPLETE. ERRORS: 0
```

```
 PAGE   2 [2.0] 11/4/82 16:16:33 IOLIB IOCOMASM - COMMON ASSEMBLY ROUTINES

   3                     ********************************************************************************
   4                     *
   5                     *       COPYRIGHT (C) 1982 BY HEWLETT-PACKARD COMPANY
   6                     *
   7                     ********************************************************************************
   8                     *
   9                     *
  10                     *       IOLIB     IOCOMASM
  11                     *
  12                     *
  13                     ********************************************************************************
  14                     *
  15                     *
  16                     *
  17                     *       Library - IOLIB
  18                     *       Module  - IOCOMASM
  19                     *
  20                     *       Purpose - This set of assembly language
  21                     *                 code is intended to be used as
  22                     *                 a support module for I/O drivers
  23                     *
  24                     *       Date    - 08/18/81
  25                     *       Update  - 09/22/82
  26                     *       Release - 09/22/82
  27                     *
  28                     *
  29                     *       Source  - IOLIB:COMASM.TEXT
  30                     *       Object  - IOLIB:COMASM.CODE
  31                     *
  32                     *
  33                     ********************************************************************************
  34                     *
  35                     *
  36                     *       RELEASED
  37                     *       VERSION         2.0
  38                     *
  39                     *
  40                     ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *       PASCAL DEFINITION OF MODULE
                        *
                        ********************************************************************************
                                MNAME IOCOMASM
                                SRC MODULE IOCOMASM;
                                SRC IMPORT iodeclarations;
                                SRC EXPORT
                                SRC     FUNCTION  dma_request ( temp   : ANYPTR ) : INTEGER;
                                SRC     PROCEDURE dma_release ( temp   : ANYPTR );
                                SRC     FUNCTION  bit_set     ( v : INTEGER ;
                                SRC                             b : INTEGER ) : BOOLEAN ;
                                SRC     FUNCTION  binand      ( x : INTEGER ;
                                SRC                             y : INTEGER ) : INTEGER ;
                                SRC     FUNCTION  binior      ( x : INTEGER ;
                                SRC                             y : INTEGER ) : INTEGER ;
                                SRC     FUNCTION  bineor      ( x : INTEGER ;
                                SRC                             y : INTEGER ) : INTEGER ;
                                SRC     FUNCTION  bincmp      ( x : INTEGER ) : INTEGER ;
                                SRC END; ( IOCOMASM )
                                DEF IOCOMASM_IOCOMASM
                                DEF IOCOMASM_DMA_REQUEST
                                DEF IOCOMASM_DMA_RELEASE
                                DEF IOCOMASM_BIT_SET
                                DEF IOCOMASM_BINAND
                                DEF IOCOMASM_BINIOR
                                DEF IOCOMASM_BINEOR
                                DEF IOCOMASM_BINCMP
```



```
                        ********************************************************************************
                        *
                        *       SYMBOLS FOR EXPORT - COMMON ASSEMBLY LANGUAGE ROUTINES
                        *
                        *       THE SYMBOLS DO NOT HAVE PASCAL ENTRY
                        *       POINTS SINCE THEY ARE ONLY USED BY
                        *       ASSEMBLY LANGUAGE MODULES
                        *
                        ********************************************************************************
                                DEF  DROPDMA
                                DEF  GETDMA
                                DEF  TESTDMA
                                DEF  LOGINT
                                DEF  LOGEOT
                                DEF  STBSY
                                DEF  STCLR
                                DEF  DMA_STBSY
                                DEF  ITXFR
                                DEF  ABORT_IO
                                DEF  WAIT_TFR
                                DEF  CHECK_TFR
```

```
                        *
                        *       module initialization
                        *
            0000 0000 IOCOMASM_IOCOMASM EQU *
 00000000 4E75                  RTS
                        *
                        *       bit test
                        *
            0000 0002 IOCOMASM_BIT_SET  EQU *
 00000002 205F                  MOVEA.L (SP)+,A0        save return address
 00000004 201F                  MOVE.L  (SP)+,D0        get bit #
 00000006 221F                  MOVE.L  (SP)+,D1        get numeric value
 00000008 4202                  CLR.B   D2              clear indicator
 0000000A 0101                  BTST    D0,D1           test bit in value
 0000000C 6702                  BEQ.S   BITT_EXIT
 0000000E 7401                  MOVEQ   #1,D2           if bit set set indicator
 00000010 1E82      BITT_EXIT   MOVE.B  D2,(SP)         push result
 00000012 4ED0                  JMP     (A0)            return
                        *
                        *       binary and
                        *
            0000 0014 IOCOMASM_BINAND   EQU *
 00000014 205F                  MOVEA.L (SP)+,A0        save return address
 00000016 201F                  MOVE.L  (SP)+,D0        get last param
 00000018 221F                  MOVE.L  (SP)+,D1        get first param
 0000001A C280                  AND.L   D0,D1           perform AND
 0000001C 2E81                  MOVE.L  D1,(SP)         push result
 0000001E 4ED0                  JMP     (A0)            return
                        *
                        *       binary inclusive or
                        *
            0000 0020 IOCOMASM_BINIOR   EQU *
 00000020 205F                  MOVEA.L (SP)+,A0        save return address
 00000022 201F                  MOVE.L  (SP)+,D0        get last param
 00000024 221F                  MOVE.L  (SP)+,D1        get first param
 00000026 8280                  OR.L    D0,D1           perform OR
 00000028 2E81                  MOVE.L  D1,(SP)         push result
 0000002A 4ED0                  JMP     (A0)            return
                        *
                        *       binary exclusive or
                        *
            0000 002C IOCOMASM_BINEOR   EQU *
 0000002C 205F                  MOVEA.L (SP)+,A0        save return address
 0000002E 201F                  MOVE.L  (SP)+,D0        get last param
 00000030 221F                  MOVE.L  (SP)+,D1        get first param
 00000032 B181                  EOR.L   D0,D1           perform XOR
 00000034 2E81                  MOVE.L  D1,(SP)         push result
 00000036 4ED0                  JMP     (A0)            return
                        *
                        *       binary complement
                        *
            0000 0038 IOCOMASM_BINCMP   EQU *
 00000038 205F                  MOVEA.L (SP)+,A0        save return address
 0000003A 201F                  MOVE.L  (SP)+,D0        get param
 0000003C 4680                  NOT.L   D0              perform complement
 0000003E 2E80                  MOVE.L  D0,(SP)         push result
 00000040 4ED0                  JMP     (A0)            return
```

```
PAGE   7 [2.0] 11/4/82 16:16:33 IOLIB IOCOMASM - COMMON EQUATES AND DEFINITIONS
 156                                INCLUDE IOLIB:COMDCL
```

```
PAGE   8 [2.0] 11/4/82 16:16:33 IOLIB COMDCL - COMMON EQUATES AND DEFINITIONS
 159                    ********************************************************************************
 160                    *
 161                    *       modified:  02/22/82  JPC   added parm to user EOT & ISR proc's
 162                    *
 163                    ********************************************************************************
 164                    *
 165                    *       HPL CONVENTIONS
 166                    *
 167                    *
 168                    *         Much of this code is taken intact from the 9826 HPL
 169                    *         language system EIO ROM ( extended I/O ROM ).
 170                    *         [illegible written
 171                    *         The Pascal that [illegible] calling this code uses
 172                    *         the stack for parameter passage.  The HPL code
 173                    *         uses the Ax and Dx registers for all parameters.
 174                    *         The Pascal driver entry points on the previous pages
 175                    *         take care of getting the parameters into the correct
 176                    *         registers.
 177                    *
 178                    *
 179                    *       GENERAL HPL ENTRY/EXIT CONDITIONS:
 180                    *
 181                    *         A1.L = CARD ADDRESS
 182                    *         A2.L = DRIVER TEMP ADDRESS
 183                    *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
 184                    *
 185                    *
 186                    *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
 187                    *
 188                    *         A3.L = BUFFER CONTROL BLOCK ADDRESS
 189                    *                In addition to the A1/A2 convention, Pascal will use
 190                    *                A3 for a pointer to the buffer control block.
 191                    *                The HPL system kept much of the transfer
 192                    *                information in the s.c. temps.
 193                    *
 194                    *         TIMEOUT(A2) = contains timeout information
 195                    *                Timeout was a global temp in HPL and a timeout
 196                    *                generated an error.
 197                    *                In PASCAL each card has a timeout value stored in
 198                    *                its temporary area.  A timeout error
 199                    *                generates an ESCAPE ( which can be trapped ).
 200                    *
 201                    *
 202                    ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *          DRIVER TEMPS TEMPLATE
                        *
                        *            OFFSET FROM A2
                        *
                        *            HPL DECLARATIONS ( MODIFIED )
                        *
                        *
                        ********************************************************************************
0000 0000 ISR_ENTRY     EQU 0       ..19        PASCAL ISR LINK & UNLINK area
0000 0014 USER_ISR      EQU 20                  user ISR:  do NOT change the proc/stat link/parm ordering!!!
0000 0014 H_ISR_PR      EQU 20      ..23            procedure ptr
0000 0018 H_ISR_SL      EQU 24      ..27            static link
0000 001C H_ISR_PM      EQU 28      ..31            parameter
0000 0020 C_ADR         EQU 32      ..35        card address
0000 0024 BUFI_OFF      EQU 36      ..39        buffer pointer offset
0000 0028 BUFO_OFF      EQU 40      ..43        buffer pointer offset
0000 002C EIRB_OFF      EQU 44                  eir byte
0000 002D IO_SC         EQU 45                  select code ( i.e. 7, 22, etc. )
0000 002E TIMEOUT       EQU 46      ..49        timeout value
                        *                           =0 : no timeout
                        *                           #0 : value of timeout
0000 0032 MA_W          EQU 50      ..51        word access to my address
0000 0033 MA            EQU 51                  byte access to my address
0000 0034 AVAIL_OFF     EQU 52      ..??        standard space taken from temps
                        *           52      ..83    normal cards ( 32  bytes )
                        *           52      ..179   98628  card  ( 128 bytes )
```



```
                        ********************************************************************************
                        *
                        *       TRANSFER OFFSETS IN BUFFER CONTROL BLOCK
                        *
                        *           OFFSET FROM A3
                        *
                        *           PASCAL DECLARATION
                        *
                        ********************************************************************************
0000 0000 TTMP_OFF      EQU 0   ..3         pointer to driver temp offset
0000 0005 T_SC_OFF      EQU 5               transfer select code
0000 0007 TACT_OFF      EQU 7               actual transfer mode
0000 0009 TUSR_OFF      EQU 9               transfer mode
                        *                       00   -                  not used
                        *                       01   serial DMA
                        *                       02   serial FHS
                        *                       03   serial FASTEST ( DMA or FHS )
                        *                       04   -                  not used
                        *                       ---------------
                        *                       05   overlp INTR
                        *                       06   overlp DMA
                        *                       07   overlp FHS     ( BURST )
                        *                       08   overlp FASTEST ( DMA or BURST )
                        *                       09   overlp OVERLAP ( DMA or INTR )
0000 000A T_BW_OFF      EQU 10              transfer byte/word indicator
                        *                       0 = byte   / 1 = word
0000 000B TEND_OFF      EQU 11              transfer EOI/END indicator
                        *                       0 = no eoi / 1 = eoi sent or searched for
0000 000D TDIR_OFF      EQU 13              transfer direction
                        *                       0 = input  /  1 = output
0000 000E TCHR_OFF      EQU 14  ..15        transfer terminate character
                        *                       -1 = no termination character
0000 0010 TCNT_OFF      EQU 16  ..19        transfer count
0000 0014 TBUF_OFF      EQU 20  ..23        transfer buffer pointer
0000 0018 TBSZ_OFF      EQU 24  ..27        transfer buffer maximum size
0000 001C TEMP_OFF      EQU 28  ..31        transfer empty pointer pointer
0000 0020 TFIL_OFF      EQU 32  ..35        transfer fill pointer
0000 0024 T_PR_OFF      EQU 36  ..39        transfer pointer to eot procedure
                        *                   NIL  no procedure
0000 0028 T_SL_OFF      EQU 40  ..43        transfer eot proc static link
0000 002C T_PM_OFF      EQU 44  ..47        transfer eot proc parameter
0000 0030 T_DMAPRI      EQU 48              dma priority request
                        *
                        *       TRANSFER EQUATES
                        *
0000 0001 TT_INT        EQU 1               interrupt
0000 0002 TT_DMA        EQU 2               DMA
0000 0003 TT_BURST      EQU 3               burst
0000 0004 TT_FHS        EQU 4               fast handshake
```

```
                        ********************************************************************************
                        *
                        *       EXTERNAL REFERANCES for escape
                        *
                        ********************************************************************************
                                  REFA iodeclarations           reference the io lib var. area
                                  REFA sysglobals


                        ********************************************************************************
                        *
                        *       Escape code values
                        *
                        ********************************************************************************
              0000 0001 NO_CARD   EQU    1                     no interface
              0000 0002 NOT_HPIB  EQU    2                     not an hpib interface
              0000 0003 NO_ACTL   EQU    3                     no active controller
              0000 0004 NO_DVC    EQU    4                     sc ( not device ) specified
              0000 0005 NO_SPACE  EQU    5                     not enough space in the buffer
              0000 0006 NO_DATA   EQU    6                     not enough data left in the buffer
              0000 0007 TFR_ERR   EQU    7                     tfr error
              0000 0008 SC_BUSY   EQU    8                     sc is currently busy
              0000 0009 BUF_BUSY  EQU    9                     the buffer is busy
              0000 000A TCNTERR   EQU    10                    bad count
              0000 000B BADTMO    EQU    11                    bad timeout value
              0000 000C NO_DRV    EQU    12                    no driver
              0000 000D NO_DMA    EQU    13                    no dma installed
              0000 000E NO_WORD   EQU    14                    no word transfers allowed
              0000 000F NOT_TALK  EQU    15                    not addressed as talker
              0000 0010 NOT_LSTN  EQU    16                    not addressed as listener
              0000 0011 TMO_ERR   EQU    17                    timeout
              0000 0012 NO_SCTL   EQU    18                    not system controller
              0000 0013 BAD_RDS   EQU    19                    bad read status / write control
              0000 0014 BAD_SCT   EQU    20                    bad set/clear/test
              0000 0015 CRD_DWN   EQU    21                    interface is dead
              0000 0016 EOD_SEEN  EQU    22                    end of data has happened
              0000 0017 IO_MISC   EQU    23                    misc. error


              FFFF FFE6 IOE_ERROR EQU    -26                   io sub system error escape code


              FFFF FFBE IOE_RSLT  EQU    IODECLARATIONS-66
              FFFF FFBA IOE_SC    EQU    IODECLARATIONS-70


              FFFF FFFE ESC_CODE  EQU    SYSGLOBALS-2
              FFFF FFF6 RCVR_BLK  EQU    SYSGLOBALS-10
```



```
                        ********************************************************************************
                        *
                        *       Error escapes
                        *
                        ********************************************************************************
    00000042 7011       CTMO_ERR  MOVEQ  #TMO_ERR,D0           timeout
    00000044 600A                 BRA.S  ESC_ERR
    00000046 700A       TERR_C    MOVEQ  #TCNTERR,D0           bad transfer specification
    00000048 6006                 BRA.S  ESC_ERR
    0000004A 7007       TERR_B    MOVEQ  #TFR_ERR,D0           bad transfer specification
    0000004C 6002                 BRA.S  ESC_ERR
    0000004E 700D       TERR_D    MOVEQ  #NO_DMA,D0            DMA not installed
                        *         BRA.S  ESC_ERR


    00000050 48C0       ESC_ERR   EXT.L  D0
    00000052 2B40 FFBE            MOVE.L D0,IOE_RSLT(A5)       save io error
    00000056 102A 002D            MOVE.B IO_SC(A2),D0          \ get sc for error
    0000005A 2B40 FFBA            MOVE.L D0,IOE_SC(A5)         /
    0000005E 3B7C FFE6            MOVE.W #IOE_ERROR,ESC_CODE(A5)  give i/o error
             FFFE
    00000064 4E4A                 TRAP   #10                   escape
```

```
                        ********************************************************************************
                        *
                        *       ABORT_IO
                        *
                        *         USED DURING INITIALIZATION/RESET TO MAKE SURE THERE
                        *               IS NO ACTIVE BUFFER LEFT AROUND.
                        *
                        *         ENTRY:  A2.L = TEMP POINTER
                        *
                        *         USES:   D1,D2,D3  AND ROUTINE DROPDMA (WHICH USES A0)
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
00000066 4E4B           ABORT_IO  TRAP  #11                       GET INTO SUPERVISOR MODE, SAVE SR  scs
                        * scs     MOVE    SR,-(SP)                \ PREVENT INTERRUPTS FOR A MOMENT.
00000068 007C 2700              ORI     #$2700,SR                 /
0000006C 617C           ABORT_IO3 BSR.S   ITXFR                   IS THERE A TRANSFER IN PROGRESS?
0000006E 672E                   BEQ.S   ABORT_IO2                 IF NOT, DO NOTHING
00000070 B23C 0002              CMP.B   #TT_DMA,D1                ELSE IS IT A DMA?
00000074 671A                   BEQ.S   ABORT_IO1                 IF NOT  SKIP
00000076 6100 01D6              BSR     DROPDMA                   ELSE FREE UP THE DMA CH, GET COUNT
0000007A 2744 0010              MOVE.L  D4,TCNT_OFF(A3)           fix up count
0000007E 9684                   SUB.L   D4,D3                     fix up actual count
00000080 4A2B 000D              TST.B   TDIR_OFF(A3)
00000084 6606                   BNE.S   AB_OUT
00000086 D7AB 0020              ADD.L   D3,TFIL_OFF(A3)           if input then update fill
0000008A 6004                   BRA.S   ABORT_IO1
0000008C D7AB 001C AB_OUT       ADD.L   D3,TEMP_OFF(A3)           if output then update empty
00000090 177C 00FF ABORT_IO1    MOVE.B  #255,T_SC_OFF(A3)         UNBUSY THE BUFFER
                   0005
00000096 422B 0007              CLR.B   TACT_OFF(A3)              SET TRANSFER TYPE TO NONE
0000009A 4294                   CLR.L   (A4)                      clear buffer ptr
0000009C 60CE                   BRA     ABORT_IO3                 see if there is another
0000009E 46DF           ABORT_IO2 MOVE  (SP)+,SR                  RESTORE USER MODE                 scs
000000A0 4E75                   RTS                                                                 scs
                        * scs     RTE                             RESTORE INTERRUPT LEVEL & RETURN
```



```
                        ********************************************************************************
                        *
                        *       CHECK_TFR
                        *
                        *         ROUTINE TO CHECK FOR ACTIVE TRANSFER IN THE OPPOSITE DIRECTION.
                        *               ( this is called by a tfr routine on cards
                        *                 that can't do bi-directional tfrs )
                        *               ( gpio and hpib modules use this routine )
                        *               ( with a timeout wait )
                        *
                        *         ENTRY:  A2.L = TEMP POINTER
                        *                 A3.L = BUF CTL BLK POINTER
                        *
                        *         EXIT :  IF NOT TRANSFER, RETURN
                        *                 IF     TRANSFER, THEN wait until finished
                        *                                  or   until timeout ( if any )
                        *
                        ********************************************************************************
000000A2 4A2B 000D CHECK_TFR TST.B   TDIR_OFF(A3)          base test on direction
000000A6 6608                BNE.S   CHKT_IN               ( if this is in , check out )
000000A8 49EA 0028 CHKT_OUT  LEA     BUFO_OFF(A2),A4       IS THERE  AN output  BUFFER ACTIVE?
000000AC 2214                MOVE.L  (A4),D1
000000AE 6006                BRA.S   CHKWAIT               IF SO , THEN WAIT


000000B0 49EA 0024 CHKT_IN   LEA     BUFI_OFF(A2),A4       is there an input  tfr
000000B4 2214                MOVE.L  (A4),D1
                        *    BRA.S   CHKWAIT


000000B6 6714        CHKWAIT BEQ.S   CHKEXIT               exit if no tfr
000000B8 242A 002E           MOVE.L  TIMEOUT(A2),D2        get timeout value
000000BC 67E4                BEQ.S   CHECK_TFR             if timeout = 0 then try forever
000000BE E18A                LSL.L   #8,D2
000000C0 2214        CHKLOOP MOVE.L  (A4),D1               check the buffer again
000000C2 6708                BEQ.S   CHKEXIT               if finished in time then return
000000C4 5382                SUBQ.L  #1,D2                 decrement
000000C6 66F8                BNE.S   CHKLOOP
000000C8 6000 FF78           BRA     CTMO_ERR
000000CC 4E75        CHKEXIT RTS
```

```
                        ********************************************************************************
                        *
                        *       WAIT_TFR
                        *
                        *         ROUTINE TO CHECK FOR ACTIVE TRANSFER.
                        *               ( with a timeout wait )
                        *
                        *         ENTRY:  A2.L = TEMP POINTER
                        *
                        *         EXIT :  IF NOT TRANSFER, RETURN
                        *                 IF     TRANSFER, THEN wait until finished
                        *                                      or   until timeout ( if any )
                        *
                        *         USES:   NO REGS OTHER THAN RETURN VALUES.
                        *
                        *
                        ********************************************************************************
000000CE 6100 001A WAIT_TFR  BSR    ITXFR                   quick check for tfr
000000D2 6714                BEQ.S  WT_DONE                 and exit
000000D4 2C2A 002E           MOVE.L TIMEOUT(A2),D6          get timeout value
000000D8 67F4                BEQ.S  WAIT_TFR                if timeout = 0 then try forever
000000DA EB8E                LSL.L  #5,D6
000000DC 610C       WT_LOOP  BSR.S  ITXFR                   try
000000DE 6708                BEQ.S  WT_DONE                 if finished in time then return
000000E0 5386                SUBQ.L #1,D6                   decrement
000000E2 66F8                BNE.S  WT_LOOP
000000E4 6000 FF5C           BRA    CTMO_ERR
000000E8 4E75       WT_DONE  RTS
```



```
                        ********************************************************************************
                        *
                        *       ITXFR
                        *
                        *         ROUTINE TO CHECK FOR ACTIVE TRANSFER.
                        *
                        *         ENTRY:  A2.L = TEMP POINTER
                        *
                        *         EXIT :  IF NOT TRANSFER, RET with zero flag  set
                        *                 IF     TRANSFER, RET with not zero
                        *                    D1.W = ACTUAL TFR TYPE
                        *                    D2.W = TERMINATING CHAR FROM TEMPS
                        *                    D3.L = TRANSFER COUNT FROM TEMPS
                        *                    A0.L = DATA POINTER FROM TEMPS ( either emtpy or fill )
                        *                    A3.L = BUF CTL BLK POINTER FROM TEMPS
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
000000EA 49EA 0024 ITXFR     LEA     BUFI_OFF(A2),A4        IS THERE  AN input  BUFFER ACTIVE?
000000EE 2214                MOVE.L  (A4),D1
000000F0 6608                BNE.S   ITXFR3                 IF NOT, SKIP
000000F2 49EA 0028           LEA     BUFO_OFF(A2),A4        is there an output tfr
000000F6 2214                MOVE.L  (A4),D1
000000F8 6722                BEQ.S   ITXFR1                    -no
000000FA 2641       ITXFR3   MOVEA.L D1,A3                  \
000000FC 4281                CLR.L   D1                       ELSE GET BUFFER TYPE WORD
000000FE 122B 0007           MOVE.B  TACT_OFF(A3),D1        /
00000102 4282                CLR.L   D2
00000104 342B 000E           MOVE.W  TCHR_OFF(A3),D2        GET TERMINATING CHAR
00000108 262B 0010           MOVE.L  TCNT_OFF(A3),D3        GET COUNT
0000010C 206B 001C           MOVEA.L TEMP_OFF(A3),A0        GET EMPTY POINTER
00000110 4A2B 000D           TST.B   TDIR_OFF(A3)           check direction
00000114 6604                BNE.S   ITXFR2                 \ IF INPUT
00000116 206B 0020           MOVEA.L TFIL_OFF(A3),A0        )    THEN GET FILL POINTER
0000011A 7A01       ITXFR2   MOVEQ   #1,D5                  set not zero
                0000 011C STCLR1     EQU     *
0000011C 4E75       ITXFR1   RTS
```

```
                        ********************************************************************************
                        *
                        *       STCLR
                        *
                        *         ROUTINE TO SET A BUFFER & SELECT CODE NOT BUSY
                        *
                        *         ENTRY:  gets buf ptr from ITXFR routine
                        *
                        *                 assumes only one tfr per select code
                        *
                        *         USES:   A3,D0
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
0000011E 61CA           STCLR   BSR     ITXFR                   GET BUFFER POINTER FROM TEMPS
00000120 67FA                   BEQ.S   STCLR1                  IF ALREADY CLEAR, SKIP
00000122 177C 00FF              MOVE.B  #255,T_SC_OFF(A3)       CLEAR S.C. INDICATOR IN THE BUF CTL BLK
         0005
00000128 422B 0007              CLR.B   TACT_OFF(A3)            clear tfr type
0000012C 4294                   CLR.L   (A4)                    CLEAR BUF POINTER    IN SC TEMPS
                        *RTS
                        ********************************************************************************
                        *
                        *       LOGEOT
                        *
                        *         CALL THE USER PROC AT END OF TRANSFER
                        *
                        *         PASCAL ROUTINE
                        *
                        *         modified to pass a user parameter:  JPC  02/22/82
                        *
                        ********************************************************************************
0000012E 41EB 0024      LOGEOT  LEA     T_PR_OFF(A3),A0         point to procedure/static link/parameter

00000132 2010           H_EOT1  MOVE.L  (A0),D0                 is there a proc?
00000134 6718                   BEQ.S   H_EOT3                  skip if not

00000136 48E7 0078              MOVEM.L A1-A4,-(SP)             save dedicated regs      (8/10/82 JPC)
0000013A 2F28 0008              MOVE.L  8(A0),-(SP)             push the parameter
0000013E 2228 0004              MOVE.L  4(A0),D1                is there a static link?
00000142 6702                   BEQ.S   H_EOT2
00000144 2F01                   MOVE.L  D1,-(SP)
00000146 2040           H_EOT2  MOVEA.L D0,A0                   procedure address
00000148 4E90                   JSR     (A0)                    call it
0000014A 4CDF 1E00              MOVEM.L (SP)+,A1-A4             restore dedicated regs   (8/10/82 JPC)

0000014E 4E75           H_EOT3  RTS
```



```
                        ********************************************************************************
                        *
                        *       LOGINT
                        *
                        *         THIS ROUTINE WAS CALLED H_LOG
                        *
                        *         CALL THE USER PROC WHEN AN ISR SAYS TO
                        *
                        *         PASCAL ROUTINE
                        *
                        *         modified to pass a user parameter:  JPC  02/22/82
                        *
                        ********************************************************************************
00000150 41EA 0014      LOGINT  LEA     H_ISR_PR(A2),A0         point to procedure/static link/parameter
00000154 60DC                   BRA     H_EOT1                  call it (if it exists)
```

```
                        ********************************************************************************
                        *
                        *       DMA_STBSY
                        *
                        *         ROUTINE TO SET A BUFFER BUSY
                        *
                        *         ENTRY:
                        *                 D0.W = TRANSFER COUNT TO BE PUT IN TCNT_OFF(A2)
                        *                        AND TO BE ADDED TO E/F COUNT.
                        *                 A0.L = pointer to DMA temps
                        *                 A2.L = POINTER TO DRIVER TEMPS
                        *                 A3.L = POINTER TO BUFFER CTL BLOCK
                        *                 A4.L = POINTER TO TERMINATION ROUTINE
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
00000156 214C 0000      DMA_STBSY MOVE.L A4,DMAISR(A0)          SAVE THE TERMINATION ROUTINE
0000015A 42A8 0004              CLR.L  DMASL(A0)                CLEAR THE STATIC LINK
                        *       BRA.S  STBSY                    SET THE BUFFER BUSY
                        ********************************************************************************
                        *
                        *       STBSY
                        *
                        *         ROUTINE TO SET A BUFFER BUSY
                        *
                        *         ENTRY:
                        *                 D0.W = TRANSFER COUNT TO BE PUT IN TCNT_OFF(A2)
                        *                        AND TO BE ADDED TO E/F COUNT.
                        *                 A2.L = POINTER TO DRIVER TEMPS
                        *                 A3.L = POINTER TO BUFFER CTL BLOCK
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
0000015E 2740 0010      STBSY     MOVE.L D0,TCNT_OFF(A3)        COPY TFR COUNT INTO TEMPS.
00000162 4A2B 000D                TST.B  TDIR_OFF(A3)           \
00000166 6606                     BNE.S  STBSY1                 \
00000168 254B 0024                MOVE.L A3,BUFI_OFF(A2)          MAKE SELECT CODE BUSY
0000016C 6004                     BRA.S  STBSY2                 /
0000016E 254B 0028      STBSY1    MOVE.L A3,BUF0_OFF(A2)        /
00000172 176A 002D      STBSY2    MOVE.B IO_SC(A2),T_SC_OFF(A3) SET UP BUFFER ACTIVE SELECT CODE
                 0005
00000178 4E75                     RTS                           DONE!
```



```
                        ********************************************************************************
                        *
                        *       DMA RESOURCE MANAGEMENT ROUTINES
                        *
                        *
                        ********************************************************************************


                        ********************************************************************************
                        *
                        *       DMA RESOURCE temporaries
                        *
                        *         These resource temporaries need to be aligned with the offsets
                        *         generated by the main Pascal library.  This is not an automatic
                        *         operation - it must be done by hand if ANY new declarations are
                        *         added in the iodeclarations in front of the dma resource temps.
                        *
                        ********************************************************************************
            FFFF FFC3 DMAFLAG   EQU    iodeclarations-61      boolean indicating presence of dma hardware


            FFFF FFF8 DMA0      EQU    iodeclarations-8
            FFFF FFF8 DMAISR_0  EQU    iodeclarations-8       \
            FFFF FFFC DMASL_0   EQU    iodeclarations-4         channel 0 temps
            FFFF FFF7 DMA_SC_0  EQU    iodeclarations-9       /

            FFFF FFEE DMA1      EQU    iodeclarations-18
            FFFF FFEE DMAISR_1  EQU    iodeclarations-18      \
            FFFF FFF2 DMASL_1   EQU    iodeclarations-14        channel 1 temps
            FFFF FFED DMA_SC_1  EQU    iodeclarations-19      /

            0000 0000 DMAISR    EQU    0                      isr pointer
            0000 0004 DMASL     EQU    4                      static link
            FFFF FFFF DMA_SC    EQU    -1                     allocated s.c.


            0050 0000 DMACH0    EQU    $500000                address of dma channel 0
            0050 0008 DMACH1    EQU    $500008                address of dma channel 1
                      ********************************************************************************
                      *
                      *       ADDRESS CONSTANTS
                      *
                      ********************************************************************************
            0047 8000 H_INT_CA  EQU    $478000                ADDRESS OF INTERNAL HPIB INTERFACE
```

```
                        *
                        *       request a dma channel
                        *
            0000 017A IOCOMASM_DMA_REQUEST_EQU *
0000017A 285F               MOVEA.L (SP)+,A4            save return address
0000017C 245F               MOVEA.L (SP)+,A2            get sc temp
0000017E 226A 0020          MOVEA.L C_ADR(A2),A1        get card ptr
00000182 4E4B               TRAP  #11                   GET INTO SUPERVISOR MODE        scs
                        * scs   MOVE    SR,-(SP)        JUST IN CASE CALLER DIDN'T DISABLE
00000184 007C 2700          ORI     #$2700,SR           INTERRUPTS, I WILL.
00000188 6100 0080          BSR     TESTDMA             SEE IF DMA IS INSTALLED
0000018C 6710               BEQ.S   DR_FAIL             IF NOT, return -1
0000018E 5343               SUBQ.W  #1,D3               turn $82/$81 to $81/$80
00000190 0243 0001          ANDI.W  #1,D3               determine channel
00000194 48C3               EXT.L   D3
00000196 116A 002D          MOVE.B  IO_SC(A2),DMA_SC(A0)    ELSE CLAIM THIS CHANNEL FOR CALLER
         FFFF
0000019C 6006               BRA.S   DR_GOOD
0000019E 263C FFFF DR_FAIL  MOVE.L  #-1,D3              return -1
         FFFF
000001A4 46DF       DR_GOOD MOVE    (SP)+,SR            restore int. state
000001A6 2E83               MOVE.L  D3,(SP)             assign return value - channel ( or -1 )
000001A8 4ED4               JMP     (A4)                return addr
                        *
                        *       release a dma channel
                        *
            0000 01AA IOCOMASM_DMA_RELEASE_EQU *
000001AA 205F               MOVEA.L (SP)+,A0            save return address
000001AC 245F               MOVEA.L (SP)+,A2            get sc temp
000001AE 226A 0020          MOVEA.L C_ADR(A2),A1        get card ptr
000001B2 4850               PEA     (A0)                push return address
000001B4 6000 0098          BRA     DROPDMA             release it
```



```
                        ********************************************************************************
                        *
                        *       GETDMA
                        *
                        *         ROUTINE TO OBTAIN CONTROL OF A DMA CHANNEL
                        *         GET EITHER DMA CHANNEL, TRYING FOR CH 1 FIRST.
                        *
                        *         ENTR:   CONDITIONS ARE THE SAME AS FOR THE tfr DRIVER ENTRY POINT.
                        *
                        *         EXIT:   IF DMA IS NOT INSTALLED, 'no dma' escape is generated.
                        *                 IF DMA IS INSTALLED, THE ALGORITHM WAITS FOR A CHANNEL TO
                        *                 BECOME AVAILABLE AND THEN:
                        *                    LOGS USE OF DMA CHANNEL
                        *                    SETS UP ADDRESS AND COUNT REGISTERS.
                        *                    CONSTRUCTS CARD ARM AND DMA ARM MASKS AS FOLLOWS:
                        *                           D2.W = DMA ARM BYTE WITH BITS 1, 2 DEFINED BY
                        *                                  CONTENTS OF D1 AND BIT 0 = 1.
                        *                           D3.B = CARD ENABLE BYTE WITH BITS 0, 1 DEFINED BY
                        *                                  WHICH DMA CHANNEL WAS GRANTED AND BIT 7=1.
                        *                    A4.L = ADDRESS OF DMA CHANNEL ARM WORD.
                        *
                        *         NOTE:   IF THE REQUEST IS FOR INTERNAL HP-IB AS INDICATED BY A1,
                        *                 ONLY CHANNEL 0 WILL BE GRANTED.
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
000001B8 4E4B       GETDMA  TRAP  #11                   GET INTO SUPERVISOR MODE        scs
                        * scs   MOVE   SR,-(SP)         JUST IN CASE CALLER DIDN'T DISABLE
000001BA 007C 2700          ORI    #$2700,SR            INTERRUPTS, I WILL.
000001BE 0C80 0001          CMPI.L #$010001,D0          \ make sure count <=65536
         0001
000001C4 6A00 FE80           BPL   TERR_C               /
000001C8 6140               BSR.S  TESTDMA              SEE IF DMA IS INSTALLED
000001CA 6700 FE82           BEQ   TERR_D               IF NOT, GIVE ERROR
000001CE 116A 002D          MOVE.B IO_SC(A2),DMA_SC(A0)     ELSE CLAIM THIS CHANNEL FOR CALLER
         FFFF
000001D4 2842               MOVEA.L D2,A4               A4 = ADDRESS OF DMA CHANNEL HARDWARE.

000001D6 242B 001C          MOVE.L TEMP_OFF(A3),D2      \
000001DA 4A2B 000D          TST.B  TDIR_OFF(A3)          \
000001DE 6604               BNE.S  GETDMA1                  SET UP ADDRESS
000001E0 242B 0020          MOVE.L TFIL_OFF(A3),D2       /
000001E4 28C2       GETDMA1 MOVE.L D2,(A4)+             /

000001E6 5380               SUBQ.L #1,D0                    COUNT REGISTERS (COUNT REG
000001E8 38C0               MOVE.W D0,(A4)+                 MUST BE COUNT-1)
000001EA 5280               ADDQ.L #1,D0


000001EC 4242               CLR.W  D2
000001EE 142B 000D          MOVE.B TDIR_OFF(A3),D2      MOVE DIRECTION BIT INTO B2 OF D2
000001F2 E54A               LSL    #2,D2                IN ORDER TO CONSTRUCT DMA ARM

000001F4 4A2B 000A          TST.B  T_BW_OFF(A3)         IF BYTE TRANSFER
000001F8 6702               BEQ.S  GETDMA2              THEN SKIP
```

```
000001FA 5442                ADDQ.W #2,D2                ELSE SET BIT 1 OF DMA ARM.
000001FC 4A2B 0030 GETDMA2   TST.B  T_DMAPRI(A3)         check for dma priority requested
00000200 6702                BEQ.S  GETDMA3
00000202 5042                ADDQ.W #8,D2                  if set  then set pri bit
00000204 5242      GETDMA3   ADDQ.W #1,D2                SET BITO OF DMA ARM
00000206 46DF                MOVE   (SP)+,SR                                          scs
00000208 4E75                RTS                                                      scs
                   * scs     RTE
```

```
                   ********************************************************************************
                   *
                   *       TESTDMA
                   *
                   *         THIS ROUTINE TESTS FOR PRESENCE OF DMA HARDWARE AND WAITS FOR
                   *         A CHANNEL TO BECOME AVAILABLE.
                   *
                   *         ENTRY:  A1 = CARD ADDRESS
                   *
                   *         EXIT:   IF NO DMA IS INSTALLED, RET with zero flag set
                   *                 IF    DMA IS INSTALLED, RET with not zero set
                   *                      A0.L = ADDRESS OF DMA FLAG FOR AVAILABLE CHANNEL
                   *                      D2.L = ADDRESS OF AVAILABLE DMA CHANNEL
                   *                      D3.B = CARD ENABLE BYTE FOR AVAILABLE CHANNEL
                   *
                   *         HPL ROUTINE ( MODIFIED )
                   *
                   ********************************************************************************
0000020A 41ED FFC3 TESTDMA   LEA    DMAFLAG(A5),A0        \
0000020E 4A10                TST.B  (A0)                   DO RET 1 IF NO DMA
00000210 673A                BEQ.S  TESTDMA_C              /
00000212 B3FC 0047           CMPA.L #H_INT_CA,A1          IF THIS IS A REQUEST FOR THE INTERNAL
         8000
00000218 6716                BEQ.S  TESTDMA_A             HP-IB, THEN CAN'T TRY FOR CH 1 SO SKIP.
0000021A 41ED FFEE           LEA    DMA1(A5),A0           ELSE ASSUME WE CAN GET CH 1
0000021E 243C 0050           MOVE.L #DMACH1,D2
         0008
                   * tm      MOVEQ  #$82,D3
00000224 163C 0082           MOVE.B #$82,D3
00000228 0C28 00FF           CMPI.B #255,DMA_SC(A0)       CAN WE?
         FFFF
0000022E 671A                BEQ.S  TESTDMA_B             IF SO, THEN RET 3
00000230 41ED FFF8 TESTDMA_A LEA    DMA0(A5),A0           ELSE ASSUME WE CAN GET CH 0
00000234 243C 0050           MOVE.L #DMACH0,D2
         0000
                   * tm      MOVEQ  #$81,D3
0000023A 163C 0081           MOVE.B #$81,D3
0000023E 0C28 00FF           CMPI.B #255,DMA_SC(A0)       CAN WE?
         FFFF
00000244 6704                BEQ.S  TESTDMA_B             IF HARDWARE PRESENT BUT BUSY,same as not there
00000246 4245                CLR    D5
00000248 4E75                RTS
0000024A 7A01      TESTDMA_B MOVEQ  #1,D5                 ELSE WE GOT A CH
```



```
0000024C 4E75      TESTDMA_C RTS
```

```
                   ********************************************************************************
                   *
                   *       DROPDMA
                   *
                   *         ROUTINE TO FREE UP A DMA CHANNEL
                   *
                   *         ENTRY:  A2.L = POINTER TO DRIVER TEMPS
                   *
                   *         EXIT:   D4.W = FINAL DMA CHANNEL COUNT
                   *                 CHANNEL IS DISARMED.
                   *
                   *         USES:   A0
                   *
                   *         HPL ROUTINE ( MODIFIED )
                   *
                   ********************************************************************************
         0000 024E DROPDMA   EQU *                                                   scs
0000024E 4E4B                TRAP #11
                   * scs     MOVE   SR,-(SP)              JUST IN CASE CALLER DIDN'T DISABLE
00000250 007C 2700           ORI    #$2700,SR             INTERRUPTS, I WILL.
00000254 7800                MOVEQ  #0,D4                 ASSUME DMA CHA ALREADY DROPPED...
00000256 102D FFF7           MOVE.B DMA_SC_0(A5),D0       \
0000025A B02A 002D           CMP.B  IO_SC(A2),D0          / IS IT CH 0?
0000025E 660C                BNE.S  DROPDMA0              IF NOT, SKIP
00000260 49ED FFF8           LEA    DMA0(A5),A4           GET A POINTER TO THE CHANNEL R/W
00000264 41F9 0050           LEA    DMACH0,A0             POINT A0 TO CH 0
         0000
0000026A 6014                BRA.S  DROPDMA1              GO DO IT

0000026C 102D FFED DROPDMA0  MOVE.B DMA_SC_1(A5),D0       \
00000270 B02A 002D           CMP.B  IO_SC(A2),D0           / IS IT CH 1?
00000274 6622                BNE.S  DROPDMA2              IF NOT, DO NOTHING
00000276 49ED FFEE           LEA    DMA1(A5),A4           GET A POINTER TO THE CHANNEL R/W
0000027A 41F9 0050           LEA    DMACH1,A0             POINT A0 TO CH 1
         0008
00000280 197C 00FF DROPDMA1  MOVE.B #255,DMA_SC(A4)       clear s.c.
         FFFF
00000286 42AC 0004           CLR.L  DMASL(A4)             clear static link
0000028A 42AC 0000           CLR.L  DMAISR(A4)            clear isr pointer
0000028E 2818                MOVE.L (A0)+,D4              DISARM CH BY READING ADDRESS
00000290 4284                CLR.L  D4
00000292 3810                MOVE.W (A0),D4               GET FINAL COUNT INTO D0
00000294 D87C 0001           ADD.W  #1,D4                 FIX UP COUNT TO INDICATE LEFT OVER TFR'S
00000298 46DF      DROPDMA2  MOVE   (SP)+,SR                                        scs
0000029A 4E75                RTS                                                   scs
                   * scs     RTE
```

```
  833                                   END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
IODECLARATIONS	ABS	290	00000002
SYSGLOBALS	ABS	291	00000006

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ABORT_IO	REL	368		00000066
ABORT_IO1	REL	383		00000090
ABORT_IO2	REL	387		0000009E
ABORT_IO3	REL	371		0000006C
AB_OUT	REL	382		0000008C
AVAIL_OFF	ABS	230		00000034
BADTMO	ABS	308		0000000B
BAD_RDS	ABS	316		00000013
BAD_SCT	ABS	317		00000014
BITT_EXIT	REL	112		00000010
BUFI_OFF	ABS	221		00000024
BUFO_OFF	ABS	222		00000028
BUF_BUSY	ABS	306		00000009
CCR	STREG	0		00000005
CHECK_TFR	REL	409		000000A2
CHKEXIT	REL	428		000000CC
CHKLOOP	REL	423		000000C0
CHKT_IN	REL	415		000000B0
CHKT_OUT	REL	411		000000A8
CHKWAIT	REL	419		000000B6
CRD_DWN	ABS	318		00000015
CTMO_ERR	REL	337		00000042
C_ADR	ABS	220		00000020
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DMA0	ABS	629	IODECLARATIONS +	FFFFFFF8
DMA1	ABS	634	IODECLARATIONS +	FFFFFFEE
DMACH0	ABS	643		00500000
DMACH1	ABS	644		00500008
DMAFLAG	ABS	627	IODECLARATIONS +	FFFFFFC3
DMAISR	ABS	639		00000000

```
DMAISR_0          ABS       630 IODECLARATIONS  +      FFFFFFF8
DMAISR_1          ABS       635 IODECLARATIONS  +      FFFFFFEE
DMASL             ABS       640                        00000004
DMASL_0           ABS       631 IODECLARATIONS  +      FFFFFFFC
DMASL_1           ABS       636 IODECLARATIONS  +      FFFFFFF2
DMA_SC            ABS       641                        FFFFFFFF
DMA_SC_0          ABS       632 IODECLARATIONS  +      FFFFFFF7
DMA_SC_1          ABS       637 IODECLARATIONS  +      FFFFFFED
DMA_STBSY         REL       581                        00000156
DROPDMA           REL       805                        0000024E
DROPDMA0          REL       817                        0000026C
DROPDMA1          REL       822                        00000280
DROPDMA2          REL       829                        00000298
DR_FAIL           REL       670                        0000019E
DR_GOOD           REL       671                        000001A4
EIRB_OFF          ABS       223                        0000002C
EOD_SEEN          ABS       319                        00000016
ESC_CODE          ABS       327 SYSGLOBALS      +      FFFFFFFE
ESC_ERR           REL       346                        00000050
GETDMA            REL       714                        000001B8
GETDMA1           REL       728                        000001E4
GETDMA2           REL       741                        000001FC
GETDMA3           REL       744                        00000204
H_EOT1            REL       533                        00000132
H_EOT2            REL       541                        00000146
H_EOT3            REL       545                        0000014E
H_INT_CA          ABS       650                        00478000
H_ISR_PM          ABS       219                        0000001C
H_ISR_PR          ABS       217                        00000014
H_ISR_SL          ABS       218                        00000018
IOCOMASM_BINAND   REL       117                        00000014
IOCOMASM_BINCMP   REL       147                        00000038
IOCOMASM_BINEOR   REL       137                        0000002C
IOCOMASM_BINIOR   REL       127                        00000020
IOCOMASM_BIT_SET  REL       104                        00000002
IOCOMASM_DMA_RELEASE REL    678                        000001AA
IOCOMASM_DMA_REQUEST REL    656                        0000017A
IOCOMASM_IOCOMASM REL        99                        00000000
IOE_ERROR         ABS       322                        FFFFFFE6
IOE_RSLT          ABS       324 IODECLARATIONS  +      FFFFFFBE
IOE_SC            ABS       325 IODECLARATIONS  +      FFFFFFBA
IO_MISC           ABS       320                        00000017
IO_SC             ABS       224                        0000002D
ISR_ENTRY         ABS       215                        00000000
ITXFR             REL       478                        000000EA
ITXFR1            REL       496                        0000011C
ITXFR2            REL       494                        0000011A
ITXFR3            REL       484                        000000FA
LOGEOT            REL       531                        0000012E
LOGINT            REL       561                        00000150
MA                ABS       229                        00000033
MA_W              ABS       228                        00000032
NOT_HPIB          ABS       299                        00000002
NOT_LSTN          ABS       313                        00000010
NOT_TALK          ABS       312                        0000000F
NO_ACTL           ABS       300                        00000003
NO_CARD           ABS       298                        00000001
```



```
NO_DATA           ABS       303                        00000006
NO_DMA            ABS       310                        0000000D
NO_DRV            ABS       309                        0000000C
NO_DVC            ABS       301                        00000004
NO_SCTL           ABS       315                        00000012
NO_SPACE          ABS       302                        00000005
NO_WORD           ABS       311                        0000000E
RCVR_BLK          ABS       328 SYSGLOBALS      +      FFFFFFF6
SC_BUSY           ABS       305                        00000008
SP                AREG        0                        00000007
SR                STREG       0                        00000006
STBSY             REL       600                        0000015E
STBSY1            REL       605                        0000016E
STBSY2            REL       606                        00000172
STCLR             REL       513                        0000011E
STCLR1            REL       495                        0000011C
TACT_OFF          ABS       245                        00000007
TBSZ_OFF          ABS       268                        00000018
TBUF_OFF          ABS       267                        00000014
TCHR_OFF          ABS       264                        0000000E
TCNTERR           ABS       307                        0000000A
TCNT_OFF          ABS       266                        00000010
TDIR_OFF          ABS       262                        0000000D
TEMP_OFF          ABS       269                        0000001C
TEND_OFF          ABS       260                        0000000B
TERR_B            REL       341                        0000004A
TERR_C            REL       339                        00000046
TERR_D            REL       343                        0000004E
TESTDMA           REL       767                        0000020A
TESTDMA_A         REL       778                        00000230
TESTDMA_B         REL       788                        0000024A
TESTDMA_C         REL       787                        0000024C
TFIL_OFF          ABS       270                        00000020
TFR_ERR           ABS       304                        00000007
TIMEOUT           ABS       225                        0000002E
TMO_ERR           ABS       314                        00000011
TTMP_OFF          ABS       243                        00000000
TT_BURST          ABS       281                        00000003
TT_DMA            ABS       280                        00000002
TT_FHS            ABS       282                        00000004
TT_INT            ABS       279                        00000001
TUSR_OFF          ABS       246                        00000009
T_BW_OFF          ABS       258                        0000000A
T_DMAPRI          ABS       275                        00000030
T_PM_OFF          ABS       274                        0000002C
T_PR_OFF          ABS       271                        00000024
T_SC_OFF          ABS       244                        00000005
T_SL_OFF          ABS       273                        00000028
USER_ISR          ABS       216                        00000014
USP               STREG       0                        00000007
WAIT_TFR          REL       447                        000000CE
WT_DONE           REL       457                        000000E8
WT_LOOP           REL       452                        000000DC
```

DC

Purpose

DC contains assembly language low-level I/O drivers.

Usage

DC is used for the 98628 and 98629 interfaces.

Requirements

DC_DRV

PASS 1 COMPLETE. ERRORS: 0


```
                        ********************************************************************************
                        *
                        *       COPYRIGHT (C) 1982 BY HEWLETT-PACKARD COMPANY
                        *
                        ********************************************************************************
                        *
                        *
                        *       IOLIB     EXTDC
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *
                        *       Library - IOLIB
                        *
                        *       Purpose - This set of assembly language code is intended to be used as
                        *                 a PASCAL module for I/O drivers for use by the external I/O
                        *                 procedures library.
                        *
                        *
                        *       Date    - 10/26/81
                        *       Update  - 09/22/82
                        *       Release - 09/22/82
                        *
                        *
                        *       Source  - IOLIB:DC.TEXT
                        *       Object  - IOLIB:DC.CODE
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *       RELEASED
                        *       VERSION         2.0
                        *
                        *
                        ********************************************************************************
```

```
                    ************************************************
                    *                                              *
                    *                                              *
                    *       ****     ***    *****    ***          *
                    *       *   *   *   *     *     *   *         *
                    *       *   *   *   *     *     *   *         *
                    *       *   *   *****     *     *****         *
                    *       *   *   *   *     *     *   *         *
                    *       *   *   *   *     *     *   *         *
                    *       ****    *   *     *     *   *         *
                    *                                            *
                    *                                            *
                    *        ***     ***    *   *    *   *       *
                    *       *   *   *   *   ** **   ** **        *
                    *           *   *   *   * * *   * * *        *
                    *           *   *   *   *   *   *   *        *
                    *           *   *   *   *   *   *   *        *
                    *       *   *   *   *   *   *   *   *        *
                    *        ***     ***    *   *   *   *        *
                    *                                            *
                    *                                            *
                    ************************************************
                    *
                    *       dc_doc
                    *       comdcl
                    *       dc_decls
                    *       dc_buff
                    *       dc_comm
                    *       dc_inter
                    *       dc_rxbuf
                    *       dc_trans
                    *       dc_txbuf
                    *
                        sprint
                        llen    132
```



```
                    ********************************************************************************
                    *
                    *
                    *       The following lines are used to tell the LINKER/LOADER what this module
                    *       looks like in PASCAL terms.
                    *
                    *       Note that it is possible to create assembly modules that are functions.
                    *       These routines are called through an indirect pointer using the CALL
                    *       facility which does NOT permit functions.
                    *
                    *       This module is called 'EXTDC' ( upper or lower case - doesn't matter )
                    *       independent of the file name ( by use of the MNAME pseudo-op ).
                    *
                    *       All the externally used procedures are called 'EXTDC_@@@@@@@' in
                    *       this module.  If you are using assembly to access them use the
                    *       'EXTDC_@@@@@@@' name.  If you are using Pascal use the '@@@@@@@'
                    *       name.
                    *
                    ********************************************************************************
                            MNAME EXTDC
                            SRC MODULE EXTDC;
                            SRC IMPORT iodeclarations;
                            SRC EXPORT
                            SRC
                            SRC         PROCEDURE alvinit        ( temp : ANYPTR );
                            SRC         PROCEDURE alvinisr       ( temp : PISRIB );
                            SRC         PROCEDURE enter_data     ( temp : ANYPTR ;  x     : ANYPTR ;
                            SRC                                                     VAR c : INTEGER );
                            SRC         PROCEDURE output_data    ( temp : ANYPTR ;  x     : ANYPTR ;
                            SRC                                                     cnt   : INTEGER );
                            SRC         PROCEDURE output_end     ( temp : ANYPTR );
                            SRC         PROCEDURE direct_status  ( temp : ANYPTR ;  reg   : io_word;
                            SRC                                                     VAR x : io_word);
                            SRC         PROCEDURE direct_control ( temp : ANYPTR ;  reg   : io_word;
                            SRC                                                     val   : io_word );
                            SRC         PROCEDURE control_bfd    ( temp : ANYPTR ;  reg   : io_word;
                            SRC                                                     val   : io_word );
                            SRC
                            SRC         PROCEDURE start_tfr_out  ( temp : ANYPTR );
                            SRC         PROCEDURE start_tfr_in   ( temp : ANYPTR );
                            SRC END; ( of extdc )
```

```
                        ********************************************************************************
                        *
                        *       SYMBOLS FOR EXPORT AS PROCEDURE NAMES
                        *
                        ********************************************************************************
                                DEF EXTDC_EXTDC

                                DEF EXTDC_ALVINIT
                                DEF EXTDC_ALVINISR
                                DEF EXTDC_ENTER_DATA
                                DEF EXTDC_OUTPUT_DATA
                                DEF EXTDC_OUTPUT_END
                                DEF EXTDC_DIRECT_STATUS
                                DEF EXTDC_DIRECT_CONTROL
                                DEF EXTDC_CONTROL_BFD
                                DEF EXTDC_START_TFR_IN
                                DEF EXTDC_START_TFR_OUT

                        ********************************************************************************
                        *
                        *       SYMBOLS FOR IMPORT - not used currently in data comm
                        *                            if they are ever used - use a JSR to call them
                        *
                        ********************************************************************************
                        * REFA STBSY
                        * REFA STCLR
                        * REFA ITXFR
                        * REFA ABORT_IO
                        * REFA LOGINT
                        * REFA GETDMA
                        * REFA DROPDMA
                        * REFA TESTDMA
                        * REFA DMA_STBSY
                        *
                        *       change references to allow long jumps when the I/O      475 TM 9/17/82
                        *       modules get moved                                       475 TM 9/17/82
                        * LMODE STBSY
                        * LMODE STCLR
                        * LMODE ITXFR
                        * LMODE ABORT_IO
                        * LMODE LOGINT
                        * LMODE GETDMA
                        * LMODE DROPDMA
                        * LMODE TESTDMA
                        * LMODE DMA_STBSY
```



```
                        ********************************************************************************
                        *
                        *         PASCAL DRIVER ENTRY POINTS FOR GPIO CARDS
                        *
                        ********************************************************************************

                        *
                        *         MODULE initialization
                        *
          0000 0000 EXTDC_EXTDC             EQU *
 00000000 4E75                              RTS
                        *
                        *         ENTRY POINTS
                        *
 00000002 6000 00CC EXTDC_ALVINIT           BRA ALVINIT
 00000006 6000 02D2 EXTDC_ALVINISR          BRA TOP_ISR
 0000000A 6000 0814 EXTDC_ENTER_DATA        BRA ENTER_DATA
 0000000E 6000 07B2 EXTDC_OUTPUT_DATA       BRA OUTPUT_DATA
 00000012 6000 08CC EXTDC_OUTPUT_END        BRA OUTPUT_END
 00000016 6000 0154 EXTDC_DIRECT_STATUS     BRA DIRECT_STATUS
 0000001A 6000 00FE EXTDC_DIRECT_CONTROL    BRA DIRECT_CONTROL
 0000001E 6000 090E EXTDC_CONTROL_BFD       BRA CONTROL_BFD
 00000022 6000 05B6 EXTDC_START_TFR_IN      BRA START_TRANSFER_IN
 00000026 6000 05D8 EXTDC_START_TFR_OUT     BRA START_TRANSFER_OUT
```

```
193                     ************************************************
194                     *                                              *
195                     *   The following escapes are possible:        *
196                     *                                              *
197                     *       EOD_SEEN - on ENTER_DATA only,         *
198                     *          signifies termination with control  *
199                     *          block when too few bytes have been  *
200                     *          read.                               *
201                     *                                              *
202                     *       TMO_ERR - on ENTER_DATA if Rx queue    *
203                     *          was empty for too long              *
204                     *                - on OUTPUT_DATA if Tx queue  *
205                     *          goto blocked up for too long        *
206                     *                - on CONTROL_BFD if Tx control*
207                     *          queue got blocked up for too long   *
208                     *                - on OUTPUT_END if Tx control *
209                     *          queue got blocked up for too long   *
210                     *                                              *
211                     *                                              *
212                     *       CRD_DWN - on every routine if card     *
213                     *          was totally locked up for more than *
214                     *          one second.                         *
215                     *                                              *
216                     *   There are no checks for nested I/O.        *
217                     *                                              *
218                     ************************************************
219
220
221
222
223
224
225
226
227                     * module: DC_COMM
228                     ************************************************
229                     *                                              *
230                     *   procedure DIRECT_CONTROL (                 *
231                     *               var SCT:  select_code_table;   *
232                     *               REG:      1..127;              *
233                     *               VAL:      0..255  );           *
234                     *                                              *
235                     *   The ranges of REG & VAL are not checked    *
236                     *   for validity.                              *
237                     *                                              *
238                     ************************************************
239                             nosyms
```



```
241                     * module: DC_COMM
242                     ************************************************
243                     *                                              *
244                     *   procedure DIRECT_STATUS (                  *
245                     *               var SCT:  select_code_table;   *
246                     *               REG:      0..127;              *
247                     *               var VAL:  word    );         *
248                     *                                              *
249                     *   These registers are intercepted:           *
250                     *       0:  Gives value from RESET_ID          *
251                     *       1:  Returns true if hardware interrupts*
252                     *       2:  Returns 0                          *
253                     *       5:  Returns 2 bits saying state of Rx  *
254                     *           buffer                             *
255                     *       9:  Returns last ENTER TERM            *
256                     *       10: Returns last ENTER MODE            *
257                     *       11: Returns # bytes available in Tx    *
258                     *           queue, or 0 if there's not 3       *
259                     *           control block positions available  *
260                     *                                              *
261                     *   The range of REG is not checked for        *
262                     *   validity.                                  *
263                     *                                              *
264                     ************************************************
265
266
267
268
269
270
271
272
273
274
275                     * module: DC_TRANS
276                     ************************************************
277                     *                                              *
278                     *   procedure OUTPUT_DATA (                    *
279                     *               var SCT:  select_code_table;   *
280                     *               PTR:      ^ data_bytes;        *
281                     *               COUNT:    longword       );    *
282                     *                                              *
283                     *   This operation may hang waiting for space. *
284                     *                                              *
285                     ************************************************
```

```
                    * module: DC_INTER
                    ************************************************
                    *                                              *
                    *   procedure START_TRANSFER_IN (              *
                    *               var SCT: select_code_table );  *
                    *                                              *
                    *       This starts the card doing a transfer. *
                    *       The calling code must have already     *
                    *       linked the transfer block in to the    *
                    *       select_code_table structure.           *
                    *                                              *
                    ************************************************
```

```
                    * module: DC_INTER
                    ************************************************
                    *                                              *
                    *   procedure START_TRANSFER_OUT (             *
                    *               var SCT: select_code_table );  *
                    *                                              *
                    *       This starts the card doing a transfer. *
                    *       The calling code must have already     *
                    *       linked the transfer block in to the    *
                    *       select_code_table structure.           *
                    *                                              *
                    ************************************************
```



```
                    * module: DC_TRANS
                    ************************************************
                    *                                              *
                    *   procedure ENTER_DATA (                     *
                    *               var SCT:  select_code_table;   *
                    *               PTR:      ^ data_bytes;        *
                    *               var COUNT:longword       );    *
                    *                                              *
                    *  COUNT initially passes the number of bytes  *
                    *  which the upper level wants to read.  THE   *
                    *  ROUTINE DOES NOT NECESSARILY READ THIS MANY! *
                    *  Upon exit COUNT will be reflect the number  *
                    *  of data bytes entered, whether or not there *
                    *  is an escape.                               *
                    *                                              *
                    *  escape(EOD): Terminated by reaching a control*
                    *     block when not enough bytes have been    *
                    *     entered.  TERM&MODE may be read with     *
                    *     STATUS 9 and 10.                         *
                    *                                              *
                    ************************************************
```

```
                    * module: DC_TRANS
                    ************************************************
                    *                                              *
                    *   procedure OUTPUT_END (                     *
                    *               var SCT:  select_code_table );  *
                    *                                              *
                    *   Equivalent to the BASIC OUTPUT Sc;END.     *
                    *                                              *
                    *   This operation may hang waiting for space. *
                    *                                              *
                    ************************************************
```

```
359                     * module: DC_TRANS
360                     ************************************************
361                     *                                              *
362                     *   procedure CONTROL_BFD (                    *
363                     *               var SCT:  select_code_table;   *
364                     *               REG:      0..127;              *
365                     *               VAL:      0..255);             *
366                     *                                              *
367                     *   Control register 0 is intercepted and      *
368                     *   if MODE=0 no action is performed, otherwise *
369                     *   the card is reset IMMEDIATELY.             *
370                     *                                              *
371                     *   This operation may hang waiting for space. *
372                     *                                              *
373                     *   The ranges of REG & VAL are not checked    *
374                     *   for validity.                              *
375                     *                                              *
376                     ************************************************
377
378
379
380
```



```
382                     ************************************************
383                     *
384                     *   This code is required to check the validity
385                     *   of a 98628/9 card:
386                     *
387                     *   var DSDP: 0..65535;
388                     *       ATTR: 0..255;
389                     *
390                     *   if binand(readio(SC,1),60)=52 then begin
391                     *       DSDP := readio(SC,16395) * 256
392                     *               + readio(SC,16393);
393                     *       if (DSDP<32768) then begin
394                     *           ATTR := readio(SC,DSDP*2+1);
395                     *           if binand(ATTR,127)=1 then
396                     *               (*>>> CLAIM CARD! <<<*)  ;
397                     *       end;
398                     *   end;
399                     *
400                     *   Remember that the readio operations above
401                     *   can bus error!
402                     *
403                     ************************************************
404
405
406
407                     ************************************************
408                     *
409                     *   One of the arguments to all routines is a
410                     *   structure I call SCT:select_code_table.
411                     *   This is a structure allocated by the higher
412                     *   level Modcal code, the last 34 bytes of
413                     *   which I INITIALIZE AFTER MODCAL HAS
414                     *   INITIALIZED THE FRONT PART.
415                     *
416                     *   THE C_ADR FIELD MUST BE INITIALIZED BEFORE
417                     *   CALLING THIS ROUTINE:
418                     *
419                     *       procedure ALVINIT (
420                     *                   var SCT: select_code_table );
421                     *
422                     *   This routine also resets the card.  This
423                     *   routine should be used at INITIALIZE time
424                     *   but not at RESET (that's CONTROL 0;1).
425                     *
426                     ************************************************
```

```
PAGE  13 [2.0] 11/4/82 16:23:45 DC_DOC: DOCUMENTATION INFO

 428
 429                            list
```

```
PAGE  14 [2.0] 11/4/82 16:23:45 COMDCL: COMMON I/O DECLARATIONS
 432                            include COMDCL
```

```
                        ********************************************************************************
                        *
                        *       modified:  02/22/82  JPC   added parm to user EOT & ISR proc's
                        *
                        ********************************************************************************
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        *
                        ********************************************************************************
```



```
                        ********************************************************************************
                        *
                        *
                        *         DRIVER TEMPS TEMPLATE
                        *
                        *           OFFSET FROM A2
                        *
                        *           HPL DECLARATIONS ( MODIFIED )
                        *
                        *
                        ********************************************************************************
0000 0000 ISR_ENTRY     EQU 0       ..19        PASCAL ISR LINK & UNLINK area
0000 0014 USER_ISR      EQU 20                  user ISR:  do NOT change the proc/stat link/parm ordering!!!
0000 0014 H_ISR_PR      EQU 20      ..23            procedure ptr
0000 0018 H_ISR_SL      EQU 24      ..27            static link
0000 001C H_ISR_PM      EQU 28      ..31            parameter
0000 0020 C_ADR         EQU 32      ..35        card address
0000 0024 BUFI_OFF      EQU 36      ..39        buffer pointer offset
0000 0028 BUFO_OFF      EQU 40      ..43        buffer pointer offset
0000 002C EIRB_OFF      EQU 44                  eir byte
0000 002D IO_SC         EQU 45                  select code ( i.e. 7, 22, etc. )
0000 002E TIMEOUT       EQU 46      ..49        timeout value
                        *                           =0 : no timeout
                        *                           #0 : value of timeout
0000 0032 MA_W          EQU 50      ..51        word access to my address
0000 0033 MA            EQU 51                  byte access to my address
0000 0034 AVAIL_OFF     EQU 52      ..??        standard space taken from temps
                        *               52      ..83        normal cards ( 32  bytes )
                        *               52      ..179       98628  card  ( 128 bytes )
```

```
                        ********************************************************************************
                        *
                        *       TRANSFER OFFSETS IN BUFFER CONTROL BLOCK
                        *
                        *           OFFSET FROM A3
                        *
                        *           PASCAL DECLARATION
                        *
                        ********************************************************************************
0000 0000 TTMP_OFF      EQU 0    ..3          pointer to driver temp offset
0000 0005 T_SC_OFF      EQU 5                 transfer select code
0000 0007 TACT_OFF      EQU 7                 actual transfer mode
0000 0009 TUSR_OFF      EQU 9                 transfer mode
                        *                         00   -                 not used
                        *                         01   serial DMA
                        *                         02   serial FHS
                        *                         03   serial FASTEST ( DMA or FHS )
                        *                         04   -                 not used
                        *                         ---------------
                        *                         05   overlp INTR
                        *                         06   overlp DMA
                        *                         07   overlp FHS     ( BURST )
                        *                         08   overlp FASTEST ( DMA or BURST )
                        *                         09   overlp OVERLAP ( DMA or INTR )
0000 000A T_BW_OFF      EQU 10                transfer byte/word indicator
                        *                         0 = byte   / 1 = word
0000 000B TEND_OFF      EQU 11                transfer EOI/END indicator
                        *                         0 = no eoi / 1 = eoi sent or searched for
0000 000D TDIR_OFF      EQU 13                transfer direction
                        *                         0 = input  /  1 = output
0000 000E TCHR_OFF      EQU 14   ..15         transfer terminate character
                        *                         -1 = no termination character
0000 0010 TCNT_OFF      EQU 16   ..19         transfer count
0000 0014 TBUF_OFF      EQU 20   ..23         transfer buffer pointer
0000 0018 TBSZ_OFF      EQU 24   ..27         transfer buffer maximum size
0000 001C TEMP_OFF      EQU 28   ..31         transfer empty pointer pointer
0000 0020 TFIL_OFF      EQU 32   ..35         transfer fill pointer
0000 0024 T_PR_OFF      EQU 36   ..39         transfer pointer to eot procedure
                        *                     NIL  no procedure
0000 0028 T_SL_OFF      EQU 40   ..43         transfer eot proc static link
0000 002C T_PM_OFF      EQU 44   ..47         transfer eot proc parameter
0000 0030 T_DMAPRI      EQU 48                dma priority request
                        *
                        *       TRANSFER EQUATES
                        *
0000 0001 TT_INT        EQU 1                 interrupt
0000 0002 TT_DMA        EQU 2                 DMA
0000 0003 TT_BURST      EQU 3                 burst
0000 0004 TT_FHS        EQU 4                 fast handshake
```



```
                        ********************************************************************************
                        *
                        *       EXTERNAL REFERANCES for escape
                        *
                        ********************************************************************************
                                  REFA iodeclarations          reference the io lib var. area
                                  REFA sysglobals


                        ********************************************************************************
                        *
                        *       Escape code values
                        *
                        ********************************************************************************
0000 0001 NO_CARD       EQU    1                     no interface
0000 0002 NOT_HPIB      EQU    2                     not an hpib interface
0000 0003 NO_ACTL       EQU    3                     no active controller
0000 0004 NO_DVC        EQU    4                     sc ( not device ) specified
0000 0005 NO_SPACE      EQU    5                     not enough space in the buffer
0000 0006 NO_DATA       EQU    6                     not enough data left in the buffer
0000 0007 TFR_ERR       EQU    7                     tfr error
0000 0008 SC_BUSY       EQU    8                     sc is currently busy
0000 0009 BUF_BUSY      EQU    9                     the buffer is busy
0000 000A TCNTERR       EQU    10                    bad count
0000 000B BADTMO        EQU    11                    bad timeout value
0000 000C NO_DRV        EQU    12                    no driver
0000 000D NO_DMA        EQU    13                    no dma installed
0000 000E NO_WORD       EQU    14                    no word transfers allowed
0000 000F NOT_TALK      EQU    15                    not addressed as talker
0000 0010 NOT_LSTN      EQU    16                    not addressed as listener
0000 0011 TMO_ERR       EQU    17                    timeout
0000 0012 NO_SCTL       EQU    18                    not system controller
0000 0013 BAD_RDS       EQU    19                    bad read status / write control
0000 0014 BAD_SCT       EQU    20                    bad set/clear/test
0000 0015 CRD_DWN       EQU    21                    interface is dead
0000 0016 EOD_SEEN      EQU    22                    end of data has happened
0000 0017 IO_MISC       EQU    23                    misc. error


FFFF FFE6 IOE_ERROR     EQU    -26                   io sub system error escape code


FFFF FFBE IOE_RSLT      EQU    IODECLARATIONS-66
FFFF FFBA IOE_SC        EQU    IODECLARATIONS-70


FFFF FFFE ESC_CODE      EQU    SYSGLOBALS-2
FFFF FFF6 RCVR_BLK      EQU    SYSGLOBALS-10

                            list
```

```
609
610
611                     ********************************************************************************
612                     *                          Data Comm card RAM locations                        *
613                     *                        (byte offsets from base address)                      *
614
615             0000 0000 RESET_ID          equ     $000000
616             0000 0002 INT_DMA           equ     $000002
617             0000 0004 SEMAPHORE         equ     $000004
618             0000 4000 INT_COND          equ     $004000
619             0000 4002 COMMAND           equ     $004002
620             0000 4004 DATA_REG          equ     $004004
621             0000 4006 PRIMARY_ADDR      equ     $004006
622             0000 4008 DSDP              equ     $004008
623             0000 400C ERROR_CODE        equ     $00400C
624
625                     ************************** Data Structures Descriptor **************************
626
627             0000 0000 ATTRIBUTES        equ     $000000
628             0000 0002 TR_QUEUE_ADDR     equ     $000002
629             0000 0006 PRIM_0_ADDR       equ     $000006
630
631                     ******************************** Queue *************************************
632
633             0000 0000 TXENDBLOCKSPACE equ     $000000
634             0000 0001 RXDATABUFF_NUMB equ     $000001
635             0000 0004 TXBUFF          equ     $000004
636             0000 0024 RXBUFF          equ     $000024
637
638             0000 0000 CTRL_AREA         equ     $000000
639             0000 0010 DATA_AREA         equ     $000010
640
641                     ******************************** Buffer record *********************************
642
643             0000 0000 ADDR              equ     $000000
644             0000 0004 SIZE              equ     $000004
645             0000 0008 FILL              equ     $000008
646             0000 000C EMPTY             equ     $00000C
647
648                     ******************************** Control block *********************************
649
650             0000 0000 POINTER           equ     $000000
651             0000 0004 TERMFIELD         equ     $000004
652             0000 0006 MODEFIELD         equ     $000006
653             0000 0008 CTRLBLKSIZE       equ     $000008
654
655                     ******************************** select_code_table *****************************
656
657             0000 0034 ovrlaper          equ AVAIL_OFF+00 .. 1   ; word
658             0000 0036 usr0mask          equ AVAIL_OFF+02        ; byte
659             0000 0037 which_RXbuf       equ AVAIL_OFF+03        ; byte
660             0000 0038 last_enter_term equ AVAIL_OFF+04        ; byte
661             0000 0039 last_enter_mode equ AVAIL_OFF+05        ; byte
662             0000 003A intbits           equ AVAIL_OFF+06        ; 8 bits
663                     * unused                        07
664                     * The following 26 bytes are saved at interrupt
665             0000 003C term_and_mode     equ AVAIL_OFF+08 .. 09  ; Encompasses the two below
```



```
666             0000 003C term              equ AVAIL_OFF+08        ; byte
667             0000 003D mode              equ AVAIL_OFF+09        ; byte
668             0000 003E data_address      equ AVAIL_OFF+10 .. 13  ; pointer
669             0000 0042 data_number       equ AVAIL_OFF+14 .. 17  ; integer
670             0000 0046 outer_tx_count    equ AVAIL_OFF+18 .. 21  ; integer
671             0000 004A timeout_counter equ AVAIL_OFF+22 .. 25  ; integer
672             0000 004E inner_counter     equ AVAIL_OFF+26 .. 29  ; integer
673             0000 0052 inner_tx_count    equ AVAIL_OFF+30 .. 33  ; integer
674                     * This is where they are saved:
675             0000 0056 int_savespace     equ AVAIL_OFF+34 .. 59
676             0000 0070 SR_image          equ AVAIL_OFF+60 .. 61  ; word
677             0000 0072 RCR_hook          equ AVAIL_OFF+62 .. 69  ; procedure
678             0000 007A err_hook          equ AVAIL_OFF+70 .. 77  ; procedure
679             0000 0082 trc_hook          equ AVAIL_OFF+78 .. 85  ; procedure
680             0000 008A bt6_hook          equ AVAIL_OFF+86 .. 93  ; procedure
681             0000 0092 bt7_hook          equ AVAIL_OFF+94 .. 101 ; procedure
682
683
684
685             0000 009A sctablebytes      equ AVAIL_OFF+102       ; size of the entire table for allocation
686
687             0000 0000 error_int         equ 0   ; Bits for interrupt
688             0000 0001 rx_int            equ 1   ; register
689             0000 0002 tx_int            equ 2
690             0000 0003 ON_INTR_int       equ 3
691             0000 0004 RC_reset_int      equ 4
692             0000 0005 trace_int         equ 5
693
694                     ********************************************************************************
695
696                         list
```

```
                        *       ****    *   *   *****   *****   *****   ****     ***
                        *       *   *   *   *   *       *       *       *   *   *   *
                        *       *   *   *   *   *       *       *       *   *   *
                        *       ****    *   *   *****   *****   *****   ****     ***
                        *       *   *   *   *   *       *       *       *  *        *
                        *       *   *   *   *   *       *       *       *   *   *   *
                        *       ****     ***    *       *       *****   *   *    ***
                        *
                        *
                        *
                        ********************************************************************************
                        *                                                                              *
                        *       routine gain_access: gets access to SEMAPHORE on card for buffer       *
                        *                ===========  utilities.  If access is not gained in a       *
                        *                             preset time, an escape is performed.             *
                        *                                                                              *
                        *       At entry:                                                              *
                        *               a3.l =  card base address ($00xx0001)                          *
                        *                                                                              *
                        *       Upon normal exit:                                                      *
                        *           If escape performed then                                           *
                        *               Timeout occurred                                               *
                        *           Otherwise                                                          *
                        *               Access was gained                                              *
                        *               SR has been set to disable all but level 7 interrupt.  The     *
                        *               SR has been pushed on the stack and GAIN_ACCESS MUST BE         *
                        *               CALLED AT THE SAME LEVEL ON THE STACK!!!T!                     *
                        *                                                                              *
                        *       This bashes d2.l.                                                      *
                        *                                                                              *
                        ********************************************************************************

            0000 002A gain_access equ *
0000002A 241F               move.l  (sp)+,d2                ; Get return address
0000002C 4E4B               trap    #11                     get into supervisor, save SR    scs
                      * scs  move    sr,-(sp)                ; Push on old SR
0000002E 2F02               move.l  d2,-(sp)                ; and push on return address
00000030 007C 2700          ori     #$2700,sr               lock out all interrupts         scs
                      * scs  move.w  4(sp),d2                ; Now get old SR into d2
                      * scs  and.w   #$F0FF,d2               ; Strip off old int level
                      * scs  or.w    #$0600,d2               ; Set interrupt level 6
                      * scs  move    d2,sr                   ; and put into SR
00000034 243C 0002          move.l  #157500,d2 [CALIBRATED 1 SEC] ; Initialize counter
         673C
0000003A 4A2B 0004 galoop   tst.b   SEMAPHORE(a3)           ; Fetch semaphore bit in bit 7 (sign)
0000003E 6A1E               bpl.s   gadone                  ; If bit 7 true then done!
00000040 5382               subq.l  #1,d2                   ; Loop for preset time
00000042 66F6               bne.s   galoop

                      *             Timed out: escape, but first...
00000044 584F               addq    #4,sp                   pop return addres               scs
00000046 46DF               move    (sp)+,sr                restore user mode               scs
                      * scs  move    4(sp),sr                ; Get old SR back
00000048 6000 026E          bra     lunched                 ; Now escape
```



```
                        ********************************************************************************
                        *                                                                              *
                        *       routine release_access: releases access to SEMAPHORE on card which     *
                        *                ==============  was previously gained with gain_access.       *
                        *                                Read the notes with the above routine to      *
                        *                                see description of stack funnies.            *
                        *                                THIS MUST BE CALLED WITH A BSR INSTRUCTION!   *
                        *                                                                              *
                        *       At entry:                                                              *
                        *               a3.l =  card base address ($00xx0001)                          *
                        *                                                                              *
                        *       Upon normal exit:                                                      *
                        *               no registers are bashed.                                       *
                        *                                                                              *
                        ********************************************************************************

            0000 004C release_access equ *
0000004C 1740 0004          move.b  d0,SEMAPHORE(a3)        ; Store don't-care into semaphore.

00000050 3F2F 0004          move.w  4(sp),-(sp)             switch SR and return address    scs
00000054 2F6F 0002          move.l  2(sp),4(sp)                                             scs
         0004
0000005A 3E9F               move.w  (sp)+,(sp)                                              scs
0000005C 4E73               rte                             restore user mode, return       scs
                      * scs  move    4(sp),sr                ; Now get old SR back
                      * scs  move.w  2(sp),4(sp)              ; Now move return address up
                      * scs  move.w  (sp),2(sp)               ; into where we will return to
                      * scs  adda.l  #2,sp                   ; & bump up by one word

0000005E 4E75       gadone  rts                             ; Now return to the return address.
```

```
                        ********************************************************************************
                        *                                                                              *
                        *       routine find_TRBUF:     Sets up pointer in a2 to point to the record   *
                        *               ===========     describing the card's TRBUFF structure.        *
                        *                                                                              *
                        *       routine find_TXBUF:     Sets up pointer in a2 to point to the record   *
                        *               ===========     describing the card's TXBUFF structure.        *
                        *                                                                              *
                        *       routine find_RXBUF:     Sets up pointer in a2 to point to the record   *
                        *               ===========     describing the card's RXBUFF structure.        *
                        *                                                                              *
                        *       At entry:                                                              *
                        *               a3.l =  card base address ($00xx0001)                          *
                        *                                                                              *
                        *       Upon exit:                                                             *
                        *               a2.l =  buffer record base address (CTRLBUFF_ADDR,             *
                        *                       CTRLBUFF_SIZE, CTRLBUFF_FILL, CTRLBUFF_EMPTY,          *
                        *                       DATABUFF_ADDR, etc).  (shifted, +1+selectcode)         *
                        *               This bashes a1, d4 and d5.                                     *
                        *                                                                              *
                        ********************************************************************************

              0000 0060 find_TRBUF equ *
00000060 7A00           moveq   #0,d5
00000062 247C 0000      movea.l #TR_QUEUE_ADDR,a2
              0002
00000068 600C           bra.s   findTR

              0000 006A find_TXBUF equ *
0000006A 7A04           moveq   #TXBUFF,d5
0000006C 6002           bra.s   find

              0000 006E find_RXBUF equ *
0000006E 7A24           moveq   #RXBUFF,d5
00000070 247C 0000 find movea.l #PRIM_0_ADDR,a2
              0006

00000076 4284       findTR  clr.l   d4
00000078 090B 4008          movep.w DSDP(a3),d4
0000007C EE5C               ror.w   #7,d4
0000007E D88B               add.l   a3,d4
00000080 2244               movea.l d4,a1                   ; a1 points to Data Struct Descriptor

00000082 4284               clr.l   d4
00000084 D3CA               adda.l  a2,a1                   ; add offset to which queue table
00000086 0909 0000          movep.w 0(a1),d4
0000008A EE5C               ror.w   #7,d4
0000008C D88B               add.l   a3,d4
0000008E D885               add.l   d5,d4
00000090 2444               movea.l d4,a2                   ; a2 points to buffer record
00000092 4E75               rts
```



```
                        ********************************************************************************
                        *                                                                              *
                        *       routine find_RX_DATA:   Sets up pointers to point to the appropriate   *
                        *               ============    receive data buffer.  This is to be used after *
                        *                               the routine find_RXBUF which sets up the       *
                        *                               pointer (in a2) to the receive control buffer  *
                        *                               descriptor record structure.                   *
                        *                                                                              *
                        *       At entry:                                                              *
                        *               a3.l =  card base address ($00xx0001)                          *
                        *               a2.l =  Buffer record base address (CTRLBUFF_ADDR,             *
                        *                       CTRLBUFF_SIZE, CTRLBUFF_FILL, CTRLBUFF_EMPTY,          *
                        *                       DATABUFF0_ADDR, etc).  (shifted, +1+selectcode)        *
                        *               a4.l =  pointer to select_code_table structure                 *
                        *                                                                              *
                        *       Upon exit:                                                             *
                        *               a1.l =  data area base address (shifted, +1+selectcode)        *
                        *               d4.l =  address of first byte PAST data area (shifted, +1+sc)  *
                        *               d5.l =  XXxxxxBUFF_SIZE (unshifted, not adjusted)              *
                        *                                                                              *
                        ********************************************************************************

              0000 0094 find_RX_DATA equ *
00000094 227C 0000          movea.l #DATA_AREA,a1
              0010
0000009A 4285               clr.l   d5                      ; compute offset for WHICH rx data
0000009C 1A2C 0037          move.b  which_RXbuf(a4),d5      ; buffer we are using
000000A0 E985               asl.l   #4,d5
000000A2 D3C5               adda.l  d5,a1

000000A4 600E               bra.s   findare                 ; Now go do the rest of it!
```

```
                        ********************************************************************************
                        *                                                                              *
                        *       routine find_DATA_AREA: Sets up pointers to point to the data buffer.  *
                        *               ==============  This is to be used in conjunction with the     *
                        *                               routines find_XXBUF which will set up the      *
                        *                               pointer (in a2) to the buffer we are using.    *
                        *                               THIS SHOULD NOT BE USED WITH THE RECEIVE       *
                        *                               BUFFER!  USE THE PREVIOUS ROUTINE INSTEAD!     *
                        *                                                                              *
                        *       routine find_CTRL_AREA: Sets up pointers to point to the ctrl buffer.  *
                        *               ==============  This is to be used in conjunction with the     *
                        *                               routines find_XXBUF which will set up the      *
                        *                               pointer (in a2) to the buffer we are using.    *
                        *                                                                              *
                        *       At entry:                                                              *
                        *               a3.l =  card base address ($00xx0001)                          *
                        *               a2.l =  Data buffer record base address (CTRLBUFF_ADDR,        *
                        *                       CTRLBUFF_SIZE, CTRLBUFF_FILL, CTRLBUFF_EMPTY,          *
                        *                       DATABUFF_ADDR, etc).  (shifted, +1+selectcode)         *
                        *                                                                              *
                        *       Upon exit:                                                             *
                        *               a1.l =  data area base address (shifted, +1+selectcode)        *
                        *               d4.l =  address of first byte PAST data area (shifted, +1+sc)  *
                        *               d5.l =  XXxxxxBUFF_SIZE (unshifted, not adjusted)              *
                        *                                                                              *
                        ********************************************************************************
                
            0000 00A6 find_DATA_AREA equ *
000000A6 227C 0000          movea.l #DATA_AREA,a1
            0010
000000AC 6006               bra.s   findare
                
            0000 00AE find_CTRL_AREA equ *
000000AE 227C 0000          movea.l #CTRL_AREA,a1
            0000
                
000000B4 D3CA       findare adda.l  a2,a1                   ; a1 points to data/ctrl part of record
000000B6 4285               clr.l   d5
000000B8 0B09 0004          movep.w SIZE(a1),d5
000000BC E05D               ror.w   #8,d5                   ; d5 = SIZE in bytes
                
000000BE 4284               clr.l   d4
000000C0 0909 0000          movep.w ADDR(a1),d4
000000C4 EE5C               ror.w   #7,d4
000000C6 D88B               add.l   a3,d4
000000C8 2244               movea.l d4,a1                   ; a1 points to front of buffer area
000000CA D885               add.l   d5,d4
000000CC D885               add.l   d5,d4                   ; d4 points past end of buffer area
                
000000CE 4E75               rts
```



```
                
                        *        ***     ***    *   *    *   *     ***    *   *   ****
                        *       *   *   *   *   ** **   ** **    *   *   **  *   *   *
                        *       *       *   *   * * *   * * *    *   *   * * *   *   *
                        *       *       *   *   *   *   *   *    *****   *  **   *   *
                        *       *   *   *   *   *   *   *   *    *   *   *   *   *   *
                        *       *   *   *   *   *   *   *   *    *   *   *   *   *   *
                        *        ***     ***    *   *   *   *    *   *   *   *   ****
                
                
                
                        ********************************************************************************
                        *                                                                              *
                        *   procedure ALVINIT(var SCT: select_code_table)                              *
                        *                                                                              *
                        ********************************************************************************
                
            0000 00D0 alvinit equ     *
000000D0 205F               movea.l (sp)+,a0
000000D2 285F               movea.l (sp)+,a4
000000D4 4850               pea     (a0)
000000D6 266C 0020          movea.l c_adr(a4),a3
000000DA 528B               addq.l  #1,a3
                
000000DC 243C 0000          move.l  #sctablebytes-1-AVAIL_OFF,d2
            0065
000000E2 4234 2034 clrloop clr.b   AVAIL_OFF(a4,d2)
000000E6 51CA FFFA          dbf     d2,clrloop
                
000000EA 102B 0002          move.b  INT_DMA(a3),d0  ; This SR is used to
000000EE 0240 0030          andi.w  #$0030,d0       ; set the interrupt level
000000F2 0640 0230          addi.w  #$0230,d0       ; equal to that of the
000000F6 E940               asl.w   #4,d0           ; card.
000000F8 3940 0070          move.w  d0,SR_image(a4)
                
000000FC 422C 0037          CLR.B   WHICH_RXBUF(A4) ;                               ( SPRyyy TM 6/15/82 )
                
00000100 6100 022C          bsr     chk_err         ; See if card is giving
                        *                           ; overlapped error
00000104 6100 0176          bsr     do_reset        ; This can escape if card bad
                
00000108 6000 0192          bra     check_ov_error  ; Escape if any error
```

```
                        ********************************************************************************
                        *                                                                              *
                        *       routine eir:    Enables interrupts on this card only.                  *
                        *                       ===                                                    *
                        *                                                                              *
                        *       At entry:                                                              *
                        *               a3.l =  card's base address ($00xx0001)                        *
                        *                                                                              *
                        *       Upon normal exit:                                                      *
                        *               No registers are changed.                                      *
                        *                                                                              *
                        ********************************************************************************

            0000 010C eir       equ     *
0000010C 177C 0080              move.b  #$80,INT_DMA(a3)                ; Set enable-interrupt bit true
         0002
00000112 4E75                   rts


                        ********************************************************************************
                        *                                                                              *
                        *       routine dir:    Disables interrupts on this card only.                 *
                        *                       ===                                                    *
                        *                                                                              *
                        *       At entry:                                                              *
                        *               a3.l =  card's base address ($00xx0001)                        *
                        *                                                                              *
                        *       Upon normal exit:                                                      *
                        *               No registers are changed.                                      *
                        *                                                                              *
                        ********************************************************************************

            0000 0114 dir       equ     *
00000114 422B 0002              clr.b   INT_DMA(a3)                     ; Clear enable-interrupt bit
00000118 4E75                   rts
```



```
                        ************************************************
                        *                                              *
                        *   procedure DIRECT_CONTROL (                 *
                        *               var SCT:  select_code_table;   *
                        *               REG:      1..127 & 256 & 257;  *
                        *               VAL:      0..255  );           *
                        *                                              *
                        ************************************************

            0000 011A direct_control equ *
0000011A 205F                   movea.l (sp)+,a0
0000011C 321F                   move.w  (sp)+,d1        ; VAL
0000011E 341F                   move.w  (sp)+,d2        ; REG
00000120 285F                   movea.l (sp)+,a4        ; SCT
00000122 4850                   pea     (a0)
00000124 266C 0020              movea.l c_adr(a4),a3
00000128 528B                   addq.l  #1,a3

0000012A 6100 0170              bsr     check_ov_error  ; Escape if any error

0000012E 1741 4004              move.b  d1,DATA_REG(a3) ; Send VAL
00000132 1742 4002              move.b  d2,COMMAND(a3)  ; then REG
00000136 B43C 007D              cmp.b   #125,d2
0000013A 6608                   bne.s   not125
0000013C 6100 0422              bsr     outxfr_done     ; For ctrl 125 (abort)
00000140 6100 0430              bsr     inxfr_done      ; abort transfers
            0000 0144 not125    equ     *                       (tm) mod - 12/02/81
                        *
00000144 B47C 0100              CMP.W   #256,D2         (tm) CHECK FOR ABORT TFR IN
00000148 6700 0428              BEQ     INXFR_DONE      (tm)
0000014C B47C 0101              CMP.W   #257,D2         (tm) CHECK FOR ABORT TFR OUT
00000150 6700 040E              BEQ     OUTXFR_DONE     (tm)

00000154 203C 0002              move.l  #181851,d0 [CALIBRATED 1 SEC]   ; Now start counter for timeout
         C65B
0000015A 4A2B 4002 ctloop       tst.b   COMMAND(a3)
0000015E 6708                   beq.s   ctldun          ; Done when COMMAND=0
00000160 5380                   subq.l  #1,d0
00000162 66F6                   bne.s   ctloop                          ; Otherwise decrement counter
00000164 6000 0152              bra     lunched                         ; & escape

00000168 6000 0132 ctldun       bra     check_ov_error
```

```
1040                        ************************************************
1041                        *                                              *
1042                        *   procedure DIRECT_STATUS (                  *
1043                        *               var SCT:  select_code_table;   *
1044                        *               REG:      1..127;              *
1045                        *               var VAL:  word    );           *
1046                        *                                              *
1047                        *   These registers are intercepted:           *
1048                        *       0:  Gives value from RESET_ID          *
1049                        *       1:  Returns true if hardware interrupts *
1050                        *       2:  Returns bit 2 = in xfr active;     *
1051                        *                   bit 3 = out xfr active     *
1052                        *       5:  Returns 2 bits saying state of Rx  *
1053                        *           buffer                             *
1054                        *       9:  Returns last ENTER TERM            *
1055                        *       10: Returns last ENTER MODE            *
1056                        *       11: Returns # bytes available in Tx    *
1057                        *           queue, or 0 if there's not 3       *
1058                        *           control block positions available  *
1059                        *                                              *
1060                        ************************************************
1061
1062             0000 016C direct_status equ *
1063    0000016C 205F              movea.l (sp)+,a0
1064    0000016E 225F              movea.l (sp)+,a1        ; addr(VAL)
1065    00000170 301F              move.w  (sp)+,d0        ; REG
1066    00000172 285F              movea.l (sp)+,a4        ; SCT
1067    00000174 4850              pea     (a0)
1068    00000176 266C 0020         movea.l c_adr(a4),a3
1069    0000017A 528B              addq.l  #1,a3
1070
1071    0000017C 6100 011E         bsr     check_ov_error  ; Escape if any error
1072
1073    00000180 4241              clr.w   d1              ; d1 will hold result
1074    00000182 2F09              move.l  a1,-(sp)
1075
1076    00000184 4A00              tst.b   d0
1077    00000186 6738              beq.s   sts0
1078    00000188 0C00 0002         cmpi.b  #2,d0           ; Check for intercepted regs
1079    0000018C 6D38              blt.s   sts1
1080    0000018E 6744              beq.s   sts2
1081    00000190 0C00 0005         cmpi.b  #5,d0
1082    00000194 6758              beq.s   sts5
1083    00000196 0C00 0009         cmpi.b  #9,d0
1084    0000019A 6766              beq.s   sts9
1085    0000019C 0C00 000A         cmpi.b  #10,d0
1086    000001A0 6766              beq.s   sts10
1087    000001A2 0C00 000B         cmpi.b  #11,d0
1088    000001A6 6766              beq.s   sts11
1089    000001A8 D03C 0080         add.b   #128,d0
1090    000001AC 1940 003C         move.b  d0,term(a4)     ; Send TERM
1091
1092    000001B0 6100 0092         bsr     direct_command
1093    000001B4 122C 003D         move.b  mode(a4),d1
1094
1095    000001B8 225F       gotsts movea.l (sp)+,a1
1096    000001BA 3281              move.w  d1,(a1)                 ; Return value
```



```
1097    000001BC 6000 00DE         bra     check_ov_error
1098
1099                        *
1100                        *  Special intercepted registers
1101                        *
1102
1103    000001C0 122B 0000 sts0    move.b  RESET_ID(a3),d1
1104    000001C4 60F2              bra.s   gotsts
1105
1106    000001C6 082B 0007 sts1    btst    #7,INT_DMA(a3)
             0002
1107    000001CC 56C1              sne     d1
1108    000001CE C27C 0001         and.w   #$0001,d1
1109    000001D2 60E4              bra.s   gotsts
1110
1111    000001D4 246C 0024 sts2    movea.l BUFI_OFF(a4),a2
1112    000001D8 200A              move.l  a2,d0
1113    000001DA 6704              beq.s   sts2a
1114    000001DC 08C1 0002         bset    #2,d1
1115    000001E0 246C 0028 sts2a   movea.l BUFO_OFF(a4),a2
1116    000001E4 200A              move.l  a2,d0
1117    000001E6 67D0              beq.s   gotsts
1118    000001E8 08C1 0003         bset    #3,d1
1119    000001EC 60CA              bra.s   gotsts
1120
1121    000001EE 6100 FE7E sts5    bsr     find_RXBUF
1122    000001F2 6100 FF20         bsr     dir
1123    000001F6 6100 052A         bsr     RX_stuff_avail
1124    000001FA 3200              move.w  d0,d1
1125    000001FC 6100 FF0E         bsr     eir
1126    00000200 60B6              bra.s   gotsts
1127
1128    00000202 122C 0038 sts9    move.b  last_enter_term(a4),d1
1129    00000206 60B0              bra.s   gotsts
1130
1131    00000208 122C 0039 sts10   move.b  last_enter_mode(a4),d1
1132    0000020C 60AA              bra.s   gotsts
1133
1134    0000020E 6100 FE5A sts11   bsr     find_TXBUF
1135    00000212 6100 FF00         bsr     dir
1136    00000216 6100 FE96         bsr     find_CTRL_AREA
1137    0000021A 6100 074A         bsr     TXCTRLBUFFroom
1138    0000021E 4241              clr.w   d1
1139    00000220 0483 0000         subi.l  #12,d3
             000C
1140    00000226 6D90              blt.s   gotsts
1141    00000228 6100 FE7C         bsr     find_DATA_AREA
1142    0000022C 6100 074A         bsr     TXDATABUFFroom
1143    00000230 3203              move.w  d3,d1
1144    00000232 6100 FED8         bsr     eir
1145    00000236 6080              bra     gotsts
```

```
1147                    ********************************************************************************
1148                    *                                                                              *
1149                    *       routine put_INTMASK:    Sends the value in usr0mask to the card.       *
1150                    *                               ===========                                    *
1151                    *                                                                              *
1152                    *       At entry:                                                              *
1153                    *               a3.l =  card base address ($00xx0001)                          *
1154                    *               a4.l =  address of sc_subtabletype structure                   *
1155                    *               sc_subtabletype.usr0mask has value to send to card.            *
1156                    *                                                                              *
1157                    *       This bashes term and mode in the select code subtable.                 *
1158                    *                                                                              *
1159                    *   This bashes d0.                                                            *
1160                    *                                                                              *
1161                    ********************************************************************************
1162
1163            0000 0238 put_INTMASK equ *
1164    00000238 197C 0079          move.b  #121,term(a4)           ; Send the new driver interrupt mask
                 003C
1165    0000023E 196C 0036          move.b  usr0mask(a4),mode(a4)   ; down with control #121
                 003D
1166                    ***     bra     direct_command
1167
1168                    ********************************************************************************
1169                    *                                                                              *
1170                    *   routine direct_command                                                     *
1171                    *           ==============                                                     *
1172                    *                                                                              *
1173                    *       Uses:   a3.l =  Base address of card                                   *
1174                    *               a4.l =  Address of SCT: select_code_table                      *
1175                    *                                                                              *
1176                    *   This bashes d0.                                                            *
1177                    *                                                                              *
1178                    ********************************************************************************
1179
1180            0000 0244 direct_command equ *
1181    00000244 176C 003D          move.b  mode(a4),DATA_REG(a3)
                 4004
1182    0000024A 176C 003C          move.b  term(a4),COMMAND(a3)    ; Send TERM
                 4002
1183
1184    00000250 203C 0002          move.l  #181851,d0 [CALIBRATED 1 SEC]   ; Now start counter for timeout
                 C65B
1185    00000256 4A2B 4002 dcloop   tst.b   COMMAND(a3)
1186    0000025A 6708               beq.s   dcdone                          ; Done when COMMAND=0
1187    0000025C 5380               subq.l  #1,d0
1188    0000025E 66F6               bne.s   dcloop                          ; Otherwise decrement counter
1189    00000260 6000 0056          bra     lunched                          ; & escape
1190
1191    00000264 196B 4004 dcdone   move.b  DATA_REG(a3),mode(a4)
                 003D
1192    0000026A 4E75               rts
```



```
1194                    ********************************************************************************
1195                    *                                                                              *
1196                    *       routine get_INTMASK:    Reads the value of usr0mask from the card.     *
1197                    *                               ===========                                    *
1198                    *                                                                              *
1199                    *       At entry:                                                              *
1200                    *               a3.l =  card base address ($00xx0001)                          *
1201                    *               a4.l =  address of sc_subtabletype structure                   *
1202                    *                                                                              *
1203                    *       At exit:                                                               *
1204                    *               sc_subtabletype.usr0mask has value from the card.              *
1205                    *                                                                              *
1206                    *       This bashes term and mode in the select code subtable.                 *
1207                    *                                                                              *
1208                    *   This bashes d0.                                                            *
1209                    *                                                                              *
1210                    ********************************************************************************
1211
1212            0000 026C get_INTMASK equ *
1213    0000026C 197C 00F9          move.b  #121+128,term(a4)       ; Get the current interrupt mask
                 003C
1214    00000272 61D0               bsr     direct_command
1215    00000274 196C 003D          move.b  mode(a4),usr0mask(a4)   ; from register #121
                 0036
1216    0000027A 4E75               rts
1217
1218
1219                    ********************************************************************************
1220                    *                                                                              *
1221                    *       do_reset:  resets the card, waits for it to complete powerup and       *
1222                    *       ========   then gets INTMASK again.                                    *
1223                    *                                                                              *
1224                    *       Uses:   a3.l =  Base address of card                                   *
1225                    *               a4.l =  Address of SCT: select_code_table                      *
1226                    *                                                                              *
1227                    *   This leaves interrupts ENABLED                                             *
1228                    *                                                                              *
1229                    *   This bashes d0.                                                            *
1230                    *                                                                              *
1231                    ********************************************************************************
1232
1233
1234            0000 027C do_reset equ    *
1235    0000027C 6100 02E2          bsr     outxfr_done     ; Abort transfers
1236    00000280 6100 02F0          bsr     inxfr_done
1237    00000284 6100 FE8E          bsr     dir                     ; Disable card interrupts to prevent
1238                    *                                           ; conflicts
1239    00000288 177C 0080          move.b  #$80,RESET_ID(a3)       ; Send reset ($80) to card
                 0000
1240    0000028E 6100 FD9A          bsr     gain_access             ; Wait until SEMAPHORE is freed
1241    00000292 6100 FDB8          bsr     release_access          ; and then give it back
1242    00000296 61D4               bsr     get_INTMASK
1243    00000298 6000 FE72          bra     eir
```

```
1245                        ********************************************************************************
1246                        *                                                                              *
1247                        *   routine check_ov_error:  If the 'ovrlaper' location is nonzero then        *
1248                        *            ==============  this escapes with that error.                    *
1249                        *                                                                              *
1250                        *       Uses:   a3.l =  Base address of card ($00xx0001)                       *
1251                        *               a4.l =  Address of SCT: select_code_table                      *
1252                        *                                                                              *
1253                        *   This leaves interrupts ENABLED                                             *
1254                        *                                                                              *
1255                        ********************************************************************************
1256
1257
1258             0000 029C check_ov_error equ *
1259    0000029C 6100 FE76          bsr     dir
1260    000002A0 2F00               move.l  d0,-(sp)
1261    000002A2 4280               clr.l   d0
1262    000002A4 302C 0034          move.w  ovrlaper(a4),d0
1263    000002A8 426C 0034          clr.w   ovrlaper(a4)
1264    000002AC 6100 FE5E          bsr     eir
1265    000002B0 4A40               tst.w   d0
1266    000002B2 660A               bne.s   escape
1267    000002B4 201F               move.l  (sp)+,d0
1268    000002B6 4E75               rts
1269
1270
1271                        ***************************** Escapes ****************************************
1272
1273             0000 02B8 lunched equ     *
1274    000002B8 7015               moveq   #crd_dwn,d0
1275    000002BA 6002               bra.s   escape
1276
1277             0000 02BC time_err equ    *
1278    000002BC 7011               moveq   #tmo_err,d0
1279
1280                        ********************************************************************************
1281                        *                                                                              *
1282                        *       routine escape: performs Pascal "escape" function.  Error exit         *
1283                        *               ======  number is to be passed in d0.                        *
1284                        *                                                                              *
1285                        *       Uses:   a3.l =  Base address of card                                   *
1286                        *               a4.l =  Address of SCT: select_code_table                      *
1287                        *               a5.l =  Global pointer for escape arguments                    *
1288                        *               d0.l =  Escape number                                          *
1289                        *                                                                              *
1290                        *   This leaves interrupts ENABLED                                             *
1291                        *                                                                              *
1292                        ********************************************************************************
1293
1294             0000 02BE escape  equ     *               ; Escape point for errors
1295    000002BE 2B40 FFBE          move.l  d0,IOE_RSLT(a5)
1296    000002C2 4285               clr.l   d5              ; Tim's magic escape stuff
1297    000002C4 1A2C 002D          move.b  IO_SC(a4),d5
1298    000002C8 2B45 FFBA          move.l  d5,IOE_SC(a5)
1299    000002CC 3B7C FFE6          move.w  #IOE_ERROR,ESC_CODE(a5)
                 FFFE
```



```
1300    000002D2 177C 0080          move.b  #$80,INT_DMA(a3) ;Re-enable card interrupts if off
                 0002
1301    000002D8 4E4A               trap    #10
1302
```

```
1305
1306
1307                    *       *****   *   *   *****   *****   ****    ****    *   *   ****    *****
1308                    *         *     **  *     *     *       *   *   *   *   *   *   *   *     *
1309                    *         *     * * *     *     *       *   *   *   *   *   *   *   *     *
1310                    *         *     *  **     *     ***     ****    ****    *   *   ****      *
1311                    *         *     *   *     *     *       *  *    *  *    *   *   *         *
1312                    *         *     *   *     *     *       *   *   *   *   *   *   *         *
1313                    *       *****   *   *     *     *****   *   *   *   *    ***    *         *
1314
1315
1316
1317
1318
1319                    ********************************************************************************
1320                    *                                                                              *
1321                    *                DATA COMM CARD TOP LEVEL INTERRUPT SERVICE ROUTINE            *
1322                    *                                                                              *
1323                    *       This is reached thru the softpoll table for the appropriate            *
1324                    *       interrupt level.                                                       *
1325                    *                                                                              *
1326                    *       This handles a hardware interrupt from the data comm card.  First it   *
1327                    *       enquires to find out what interrupt conditions are pending, then it    *
1328                    *       calls the appropriate routines to handle the conditions.               *
1329                    *                                                                              *
1330                    *       At entry:                                                              *
1331                    *               a5.l =  pointer to globals area                                *
1332                    *                                                                              *
1333                    *       During this routine:                                                   *
1334                    *               a3.l =  card's base address ($00xx0001)                        *
1335                    *               a4.l =  address of sc_subtabletype structure                   *
1336                    *                                                                              *
1337                    ********************************************************************************
1338
1339            0000 02DA top_isr equ     *
1340    000002DA 205F             movea.l (sp)+,a0        ; Save return addr
1341    000002DC 285F             movea.l (sp)+,a4        ; Get sc_subtabletype
1342    000002DE 4850             pea     (a0)            ; Replace ret addr
1343
1344    000002E0 266C 0020        movea.l C_ADR(a4),a3    ; Get card base addr
1345
1346    000002E4 528B             addq.l  #1,a3           ; Now a3.l = OUR base address of card
1347
1348                    **********  TRY SECTION: RECOVER IS AT END OF ISR
1349
1350    000002E6 48E7 0018        movem.l a3/a4,-(sp)
1351    000002EA 2F2D FFF6        move.l  RCVR_BLK(a5),-(sp)
1352    000002EE 2F0E             move.l  a6,-(sp)
1353    000002F0 487A 00CE        pea     recover_section
1354    000002F4 2B4F FFF6        move.l  sp,RCVR_BLK(a5)
1355
1356                    **********  END OF 'TRY' keyword
1357
1358    000002F8 396C 003C        move.w  term_and_mode(a4),int_savespace(a4)
                0056
1359    000002FE 4CEC 003F        movem.l term_and_mode+2(a4),d0-d5
                003E
```



```
1360    00000304 48EC 003F        movem.l d0-d5,int_savespace+2(a4)
                0058
1361
1362    0000030A 196B 4000        move.b  INT_COND(a3),intbits(a4) ;Get all the interrupt condition bits
                003A
1363
1364                    *   From now on, each handler routine checks for its particular interrupting
1365                    *   condition and then jumps to the next.  This is done primarily for speed.
```

```
1367                        ********************************************************************************
1368                        *                                                                              *
1369                        *       routine remcont_reset_isr:      Takes care of communicating a          *
1370                        *               =================       card's remote-control-reset to the     *
1371                        *                                       operating system                       *
1372                        *                                                                              *
1373                        ********************************************************************************
1374
1375             0000 0310 remcont_reset_isr equ *
1376    00000310 082C 0004         btst    #RC_reset_int,intbits(a4) ; Test for bit #4 for this condition
                   003A
1377    00000316 6708              beq.s   error_isr
1378    00000318 43EC 0072         lea     RCR_hook(a4),a1
1379    0000031C 6100 00DC         bsr     try_hook
1380
1381
1382
1383                        ********************************************************************************
1384                        *                                                                              *
1385                        *       routine error_isr:   Handles the communication of an error from the    *
1386                        *               =========    interface card back to BASIC, or the hook.        *
1387                        *                                                                              *
1388                        *       At entry:                                                              *
1389                        *               a3.l =  card base address ($00xx0001)                          *
1390                        *               a4.l =  address of SCT: select_code_table                      *
1391                        *                                                                              *
1392                        *       Upon normal exit:                                                      *
1393                        *                                                                              *
1394                        *       The 'chk_err' routine is used also at powerup time.                    *
1395                        *                                                                              *
1396                        ********************************************************************************
1397
1398             0000 0320 error_isr equ *
1399    00000320 082C 0000         btst    #error_int,intbits(a4)  ; Test for bit #0 for this condition
                   003A
1400    00000326 6724              beq.s   data_rx_isr
1401
1402    00000328 6100 0004         bsr     chk_err
1403    0000032C 601E              bra.s   data_rx_isr
1404
1405    0000032E 4240      chk_err clr.w   d0
1406    00000330 102B 400C         move.b  ERROR_CODE(a3),d0       ; fetch error number
1407    00000334 6714              beq.s   errdone                 ; ignore ERROR_CODE=0
1408    00000336 422B 400C         clr.b   ERROR_CODE(a3)
1409    0000033A D07C 012C         add.w   #300,d0                 ; add offset
1410
1411    0000033E 3940 0034         move.w  d0,ovrlaper(a4)
1412    00000342 43EC 007A         lea     err_hook(a4),a1
1413    00000346 6100 00B2         bsr     try_hook
1414
1415    0000034A 4E75      errdone rts
```



```
1417                        ********************************************************************************
1418                        *                                                                              *
1419                        *       routine data_rx_isr: Handles moving received data to the user's        *
1420                        *               ============  buffer for an inbound TRANSFER.                  *
1421                        *                                                                              *
1422                        ********************************************************************************
1423
1424             0000 034C data_rx_isr equ *
1425    0000034C 082C 0001         btst    #rx_int,intbits(a4)     ; Test for bit #1 for this condition
                   003A
1426    00000352 6704              beq.s   data_tx_isr
1427    00000354 6100 00CE         bsr     do_inxfr
1428
1429                        ********************************************************************************
1430                        *                                                                              *
1431                        *       routine data_tx_isr: Handles moving transmit data from the user's      *
1432                        *               ============  buffer to the card for an outbound TRANSFER.      *
1433                        *                                                                              *
1434                        ********************************************************************************
1435
1436             0000 0358 data_tx_isr equ *
1437    00000358 082C 0002         btst    #tx_int,intbits(a4)     ; Test for bit #2 for this condition
                   003A
1438    0000035E 6704              beq.s   ON_INTR_isr
1439    00000360 6100 0186         bsr     do_outxfr
1440
1441                        ********************************************************************************
1442                        *                                                                              *
1443                        *       routine ON_INTR_isr: Handles the communication of an ON INTR trigger   *
1444                        *               ============  from the interface card back to BASIC.           *
1445                        *                                                                              *
1446                        ********************************************************************************
1447
1448             0000 0364 ON_INTR_isr equ *
1449    00000364 082C 0003         btst    #ON_INTR_int,intbits(a4) ; Test for bit #3 for this condition
                   003A
1450    0000036A 6700 0016         beq     trace_isr
1451
1452    0000036E 08AC 0003         bclr    #ON_INTR_int,usr0mask(a4)
                   0036
1453    00000374 6704              beq.s   OIisr1          ; If already 0 don't send again
1454    00000376 6100 FEC0         bsr     put_INTMASK
1455
1456    0000037A 43EC 0014 OIisr1  lea     USER_ISR(a4),a1
1457    0000037E 6100 0094         bsr     try_hook_P              (TM)    7/30/82 bug 158
1458
1459             0000 0382 trace_isr equ *
1460    00000382 082C 0005         btst    #trace_int,intbits(a4)  ; Test for bit #5 for this condition
                   003A
1461    00000388 6708              beq.s   bit_6_isr
1462    0000038A 43EC 0082         lea     trc_hook(a4),a1
1463    0000038E 6100 006A         bsr     try_hook
1464
1465             0000 0392 bit_6_isr equ *
1466    00000392 082C 0006         btst    #6,intbits(a4)          ; Test for bit #6 for this condition
                   003A
1467    00000398 6708              beq.s   bit_7_isr
```

```
1468    0000039A 43EC 008A          lea     bt6_hook(a4),a1
1469    0000039E 6100 005A          bsr     try_hook
```



```
1471              0000 03A2 bit_7_isr equ *
1472    000003A2 082C 0007          btst    #7,intbits(a4)          ; Test for bit #7 for this condition
                  003A
1473    000003A8 6708               beq.s   end_isr
1474    000003AA 43EC 0092          lea     bt7_hook(a4),a1
1475    000003AE 6100 004A          bsr     try_hook
1476
1477
1478                        * ------------------------- End of the ISR -----------------------------------
1479
1480              0000 03B2 end_isr equ     *
1481
1482                        ********** RECOVER SECTION FROM 'TRY' ABOVE *****
1483
1484    000003B2 2B6F 0008          move.l  8(sp),RCVR_BLK(a5)
                  FFF6
1485    000003B8 DFFC 0000          adda.l  #12,sp
                  000C
1486    000003BE 601E               bra.s   rcvdone
1487
1488              0000 03C0 recover_section equ *           ; On escape, flag overlapped error
1489    000003C0 2C5F               movea.l (sp)+,a6        ; to background
1490    000003C2 2B5F FFF6          move.l  (sp)+,RCVR_BLK(a5)
1491
1492                        * Body of 'RECOVER' block:
1493    000003C6 0C6D FFE6          cmpi.w  #IOE_ERROR,ESC_CODE(a5)
                  FFFE
1494    000003CC 6610               bne.s   rcvdone         ; Throw away non-I/O errors
1495    000003CE 102C 002D          move.b  IO_SC(a4),d0
1496    000003D2 B02D FFBA          cmp.b   IOE_SC(a5),d0
1497    000003D6 6606               bne.s   rcvdone
1498    000003D8 396D FFBE          move.w  IOE_RSLT(a5),ovrlaper(a4)
                  0034
1499                        * That was it!
1500
1501    000003DE 4CDF 1800 rcvdone movem.l (sp)+,a3/a4
1502    000003E2 6100 FD28          bsr     eir
1503    000003E6 396C 0056          move.w  int_savespace(a4),term_and_mode(a4)
                  003C
1504    000003EC 4CEC 003F          movem.l int_savespace+2(a4),d0-d5
                  0058
1505    000003F2 48EC 003F          movem.l d0-d5,term_and_mode+2(a4)
                  003E
1506    000003F8 4E75               rts             ; Return from ISR
1507
1508
1509    000003FA 2011       try_hook move.l (a1),d0
1510    000003FC 6714               beq.s   hook1
1511    000003FE 48E7 0018          movem.l a3/a4,-(sp)
1512    00000402 2040               movea.l d0,a0
1513    00000404 2029 0004 HOOK4    move.l  4(a1),d0                (tm)    12/03/81
1514    00000408 6702               beq.s   hook3                   (tm)    If there is static link - push it
1515    0000040A 2F00               move.l  d0,-(sp)                (tm)
1516    0000040C 4E90       hook3   jsr     (a0)
1517    0000040E 4CDF 1800 hook2    movem.l (sp)+,a3/a4
1518    00000412 4E75       hook1   rts
1519
```

```
1520
1521    00000414 2011       try_hook_P move.l (a1),d0              (TM)    7/30/82 bug 158
1522    00000416 67FA               beq.s   hook1
1523    00000418 48E7 0018          movem.l a3/a4,-(sp)
1524    0000041C 2040               movea.l d0,a0
1525    0000041E 2F29 0008          MOVE.L  8(A1),-(SP)            (TM)    7/30/82 bug 158
1526    00000422 60E0               BRA     HOOK4                  (TM)    7/30/82 bug 158
```



```
1528                        ************************************************
1529                        *                                              *
1530                        *   routine do_inxfr:  Transfers data in until: *
1531                        *               - User buffer filled;          *
1532                        *               - Card buffer empty; or        *
1533                        *               - Control block reached.       *
1534                        *                                              *
1535                        *   At entry:                                  *
1536                        *       a3.l =  card base address ($00xx0001)  *
1537                        *       a4.l =  address of select_code_table   *
1538                        *                                              *
1539                        ************************************************
1540
1541             0000 0424 do_inxfr equ    *
1542    00000424 246C 0024          movea.l BUFI_OFF(a4),a2
1543    00000428 6700 008A          beq     inxdone
1544
1545    0000042C 296A 0010          move.l  TCNT_OFF(a2),data_number(a4)
                0042
1546    00000432 6700 0090          beq     inxdn1
1547    00000436 296A 0020          move.l  TFIL_OFF(a2),data_address(a4)
                003E
1548
1549    0000043C 6100 FC30 inxfr1   bsr     find_RXBUF          ; Set up a2.l = buffer descriptor record
1550                        *                                                 base address
1551    00000440 6100 02E0 inxfr4   bsr     RX_stuff_avail      ; See if buffer is empty
1552    00000444 4A00               tst.b   d0                  ; If so, just sit here & wait
1553    00000446 6700 008C          beq     inxexit
1554
1555    0000044A 6100 0334          bsr     ctrlblknext         ; If a control block is next, then
1556    0000044E 4A00               tst.b   d0
1557    00000450 663E               bne.s   inxfr3
1558
1559    00000452 4AAC 0042          tst.l   data_number(a4) ; see if chars to transfer
1560    00000456 675C               beq.s   inxdone         ; yes, go do it
1561
1562    00000458 226C 0024          movea.l BUFI_OFF(a4),a1
1563    0000045C 3029 000E          move.w  TCHR_OFF(a1),d0 ; Check for term char desired
1564    00000460 6C06               bge.s   inxfr5          ; Yes - goto slow section
1565
1566    00000462 6100 024A          bsr     getchars        ; move some data
1567                        *                               ; & go back to check for ctrl blk
1568    00000466 60D4               bra.s   inxfr1
1569
1570             0000 0468 inxfr5   equ     *       ; SLOW ENTER - search for char
1571    00000468 1F00               move.b  d0,-(sp)        ; Save search char
1572    0000046A 2F2C 0042          move.l  data_number(a4),-(sp)
1573    0000046E 297C 0000          move.l  #1,data_number(a4)
                0001 0042
1574    00000476 6100 0236          bsr     getchars
1575    0000047A 295F 0042          move.l  (sp)+,data_number(a4)
1576    0000047E 53AC 0042          subq.l  #1,data_number(a4)
1577    00000482 101F               move.b  (sp)+,d0        ; Check for term chr
1578    00000484 206C 003E          movea.l data_address(a4),a0
1579    00000488 B028 FFFF          cmp.b   -1(a0),d0       ; If equal, exit
1580    0000048C 6726               beq.s   inxdone
1581    0000048E 60AC               bra.s   inxfr1
```

```
1582
1583    00000490 6100 01E2 inxfr3  bsr     getctrlblk      ; Get it, and check for the special
1584    00000494 0C2C 00FF         cmpi.b  #255,term(a4)   ; case TERM=255
                 003C
1585    0000049A 6608              bne.s   inxfr2
1586    0000049C 196C 003D         move.b  mode(a4),which_RXbuf(a4) ;If so, do the buffer switch
                 0037
1587    000004A2 6098              bra.s   inxfr1          ; And go back for more
1588
1589    000004A4 396C 003C inxfr2  move.w  term_and_mode(a4),last_enter_term(a4)
                 0038
1590                       *                               ; Otherwise save the control block & leave
1591    000004AA 246C 0024         movea.l BUFI_OFF(a4),a2
1592    000004AE 4A2A 000B         tst.b   TEND_OFF(a2)    ; If EOI term bit set tghen
1593    000004B2 6788              beq.s   inxfr1          ; leave else ignore
1594
1595    000004B4 246C 0024 inxdone movea.l BUFI_OFF(a4),a2
1596    000004B8 256C 003E         move.l  data_address(a4),TFIL_OFF(a2)
                 0020
1597    000004BE 256C 0042         move.l  data_number(a4),TCNT_OFF(a2)
                 0010
1598    000004C4 6100 00AC inxdn1  bsr     inxfr_done
1599    000004C8 200A              move.l  a2,d0
1600    000004CA 671A              beq.s   inxex1
1601    000004CC 43EA 0024         lea     T_PR_OFF(a2),a1
1602    000004D0 6000 FF42         bra     try_hook_P                      (TM)    7/30/82 bug 158
1603
1604              0000 04D4 inxexit equ     *
1605    000004D4 246C 0024         movea.l BUFI_OFF(a4),a2
1606    000004D8 256C 003E         move.l  data_address(a4),TFIL_OFF(a2)
                 0020
1607    000004DE 256C 0042         move.l  data_number(a4),TCNT_OFF(a2)
                 0010
1608    000004E4 67DE              beq.s   inxdn1                          BUG 1249   TM 01/08/82
1609    000004E6 4E75      inxex1  rts
```



```
1611                       ************************************************
1612                       *                                              *
1613                       *   routine do_outxfr:  Transfers data out til: *
1614                       *                - User buffer emptied; or       *
1615                       *                - Card buffer filled.           *
1616                       *                                              *
1617                       *   At entry:                                  *
1618                       *        a3.l =  card base address ($00xx0001)   *
1619                       *        a4.l =  address of select_code_table    *
1620                       *                                              *
1621                       *   During execution:                          *
1622                       *        a2.l =  address of xfr record           *
1623                       *                                              *
1624                       ************************************************
1625
1626              0000 04E8 do_outxfr equ   *
1627    000004E8 246C 0028         movea.l BUF0_OFF(a4),a2
1628    000004EC 200A              move.l  a2,d0
1629    000004EE 676E              beq.s   outxex1
1630
1631    000004F0 296A 0010         move.l  TCNT_OFF(a2),data_number(a4)
                 0042
1632    000004F6 6730              beq.s   outxd0
1633    000004F8 296A 001C         move.l  TEMP_OFF(a2),data_address(a4)
                 003E
1634
1635    000004FE 6100 FB6A         bsr     find_TXBUF      ; a2 = buff descr rec addr
1636
1637    00000502 2F2C 0042 outx1   move.l  data_number(a4),-(sp)
1638    00000506 6100 0492         bsr     putchars        ; Send some chars
1639    0000050A 201F              move.l  (sp)+,d0
1640    0000050C B0AC 0042         cmp.l   data_number(a4),d0
1641    00000510 673C              beq.s   outxit          ; If no chars transferred
1642    00000512 4AAC 0042         tst.l   data_number(a4) ; then exit
1643    00000516 66EA              bne.s   outxI           ; Check buffer empty
1644
1645    00000518 246C 0028 outxdun movea.l BUF0_OFF(a4),a2
1646    0000051C 256C 003E         move.l  data_address(a4),TEMP_OFF(a2)
                 001C
1647    00000522 256C 0042         move.l  data_number(a4),TCNT_OFF(a2)
                 0010
1648    00000528 246C 0028 outxd0  movea.l BUF0_OFF(a4),a2
1649    0000052C 4A2A 000B         tst.b   TEND_OFF(a2)    ; Check for END termination
1650    00000530 670C              beq.s   outxdn1
1651    00000532 6100 FB36         bsr     find_TXBUF      ; a2 = buff descr rec addr
1652    00000536 6100 0550         bsr     try_sending_EOF
1653    0000053A 4A00              tst.b   d0              ; If couldn't send it then
1654    0000053C 6720              beq.s   outxex1         ; return, wait for next interrupt
1655
1656    0000053E 6100 0020 outxdn1 bsr     outxfr_done
1657    00000542 200A              move.l  a2,d0
1658    00000544 6718              beq.s   outxex1
1659    00000546 43EA 0024         lea     T_PR_OFF(a2),a1
1660    0000054A 6000 FEC8         bra     try_hook_P                  (TM)    7/30/82 bug 158
1661
1662    0000054E 246C 0028 outxit  movea.l BUF0_OFF(a4),a2
```

```
1663    00000552 256C 003E          move.l  data_address(a4),TEMP_OFF(a2)
                 001C
1664    00000558 256C 0042          move.l  data_number(a4),TCNT_OFF(a2)
                 0010
1665    0000055E 4E75       outxex1 rts
```



```
1667                        ************************************************
1668                        *                                              *
1669                        *   routine outxfr_done:  Terminates the       *
1670                        *               outbound transfer, if any.     *
1671                        *                                              *
1672                        *   routine inxfr_done:  Terminates the inbound *
1673                        *               transfer, if any.              *
1674                        *                                              *
1675                        *   At entry:                                  *
1676                        *       a3.l =  card base address ($00xx0001)  *
1677                        *       a4.l =  address of select_code_table   *
1678                        *                                              *
1679                        ************************************************
1680
1681             0000 0560 outxfr_done equ *
1682    00000560 08AC 0002          bclr    #tx_int,usr0mask(a4)
                 0036
1683    00000566 6704               beq.s   outxd1          ; If already 0 don't send again
1684    00000568 6100 FCCE          bsr     put_INTMASK
1685    0000056C 41EC 0028 outxd1   lea     BUF0_OFF(a4),a0
1686    00000570 6010               bra.s   xfrdun
1687
1688
1689             0000 0572 inxfr_done equ *
1690    00000572 08AC 0001          bclr    #rx_int,usr0mask(a4)
                 0036
1691    00000578 6704               beq.s   inxd1           ; If already 0 don't send again
1692    0000057A 6100 FCBC          bsr     put_INTMASK
1693    0000057E 41EC 0024 inxd1    lea     BUF1_OFF(a4),a0
1694
1695    00000582 2450       xfrdun  movea.l (a0),a2
1696    00000584 200A               move.l  a2,d0
1697    00000586 670C               beq.s   rtsin
1698    00000588 157C 00FF          move.b  #255,T_SC_OFF(a2)
                 0005
1699    0000058E 422A 0007          clr.b   TACT_OFF(a2)
1700    00000592 4290               clr.l   (a0)
1701    00000594 4E75       rtsin   rts
```

```
1703                    ************************************************
1704                    *                                              *
1705                    *   routine wait_outxfrdone: waits until an    *
1706                    *               outbound transfer is complete  *
1707                    *               (if any).  Also has timeout    *
1708                    *               escape.                        *
1709                    *                                              *
1710                    *   routine wait_inxfrdone: waits until an     *
1711                    *               inbound transfer is complete   *
1712                    *               (if any).  Also has timeout    *
1713                    *               escape.                        *
1714                    *                                              *
1715                    *   At entry:                                  *
1716                    *       a3.l =  card base address ($00xx0001)  *
1717                    *       a4.l =  address of select_code_table   *
1718                    *                                              *
1719                    *   This bashes nothing.                       *
1720                    *                                              *
1721                    *   This routine may escape!                   *
1722                    *                                              *
1723                    *   MODIFIED  12/02/81                         *
1724                    *             from 12/01/81 code to 11/23/81   *
1725                    *                                              *
1726                    ************************************************
1727
1728          0000 0596 wait_outxfrdone equ *
1729    00000596 48E7 A080          movem.l d0/d2/a0,-(sp)
1730    0000059A 206C 0028          movea.l BUF0_OFF(a4),a0
1731    0000059E 6008               bra.s   wait
1732
1733          0000 05A0 wait_inxfrdone equ *
1734    000005A0 48E7 A080          movem.l d0/d2/a0,-(sp)
1735    000005A4 206C 0028          movea.l BUF0_OFF(a4),a0
1736
1737    000005A8 2408       wait    move.l  a0,d2           ; 0=no xfr block
1738    000005AA 6728               beq.s   waitdun
1739    000005AC 0C28 0005          cmpi.b  #5,TUSR_OFF(A0) ; >4 = interrupt
                 0009
1740    000005B2 6D20               blt.s   waitdun
1741    000005B4 242C 002E          move.l  timeout(a4),d2
1742    000005B8 203C 0000 wait1    move.l  #256,d0         [UNCALIBRATED] - guess based upon check_tfr loop
                 0100
1743    000005BE 4A28 0007 wait2    tst.b   TACT_OFF(a0)    ; 0=inactive            (tm)  tst.l -> tst.b
1744    000005C2 6710               beq.s   waitdun
1745
1746    000005C4 5380               subq.l  #1,d0           ; Timeout computation
1747    000005C6 66F6               bne.s   wait2
1748    000005C8 4A82               tst.l   d2
1749    000005CA 67EC               beq.s   wait1
1750    000005CC 5380               subq.l  #1,d0
1751    000005CE 66E8               bne.s   wait1
1752    000005D0 6000 FCEA          bra     time_err
1753
1754    000005D4 4CDF 0105 waitdun movem.l (sp)+,d0/d2/a0
1755    000005D8 4E75               rts
```



```
1757                    ************************************************
1758                    *                                              *
1759                    *   procedure START_TRANSFER_IN (              *
1760                    *               var SCT: select_code_table );  *
1761                    *                                              *
1762                    *       This starts the card doing a transfer. *
1763                    *       The calling code must have already     *
1764                    *       linked the transfer block in to the    *
1765                    *       select_code_table structure.           *
1766                    *                                              *
1767                    *   During use:                                *
1768                    *       a3.l =  card base address ($00xx0001)  *
1769                    *       a4.l =  address of select_code_table   *
1770                    *                                              *
1771                    ************************************************
1772
1773          0000 05DA START_TRANSFER_IN equ *
1774    000005DA 205F               movea.l (sp)+,a0
1775    000005DC 285F               movea.l (sp)+,a4        ; SCT
1776    000005DE 4850               pea     (a0)
1777    000005E0 266C 0020          movea.l c_adr(a4),a3
1778    000005E4 528B               addq.l  #1,a3
1779
1780    000005E6 4E4B               trap    #11             get into supervisor mode                scs
1781                    * scs       move    sr,-(sp)        ; Funny code to disable
1782    000005E8 46EC 0070          move    SR_image(a4),sr ; interrupts
1783    000005EC 08EC 0001          bset    #rx_int,usr0mask(a4)
                 0036
1784    000005F2 6100 FC44          bsr     put_INTMASK
1785    000005F6 6100 FE2C          bsr     do_inxfr
1786    000005FA 46DF               move    (sp)+,sr
1787    000005FC 6000 FC9E          bra     check_ov_error  ; Will enable ints
```

```
1789                    ************************************************
1790                    *                                              *
1791                    *   procedure START_TRANSFER_OUT (             *
1792                    *                 var SCT: select_code_table );*
1793                    *                                              *
1794                    *       This starts the card doing a transfer. *
1795                    *       The calling code must have already     *
1796                    *       linked the transfer block in to the    *
1797                    *       select_code_table structure.           *
1798                    *                                              *
1799                    *   During use:                                *
1800                    *       a3.l =  card base address ($00xx0001)  *
1801                    *       a4.l =  address of select_code_table   *
1802                    *                                              *
1803                    ************************************************
1804
1805           0000 0600 START_TRANSFER_OUT equ *
1806    00000600 205F             movea.l (sp)+,a0
1807    00000602 285F             movea.l (sp)+,a4        ; SCT
1808    00000604 4850             pea     (a0)
1809    00000606 266C 0020        movea.l c_adr(a4),a3
1810    0000060A 528B             addq.l  #1,a3
1811
1812    0000060C 4E4B             trap    #11                                                     scs
1813                    * scs     move    sr,-(sp)        ; Funny code to disable
1814    0000060E 46EC 0070        move    SR_image(a4),sr ; interrupts
1815    00000612 08EC 0002        bset    #tx_int,usr0mask(a4)
                      0036
1816    00000618 6100 FC1E        bsr     put_INTMASK
1817    0000061C 6100 FECA        bsr     do_outxfr
1818    00000620 46DF             move    (sp)+,sr
1819    00000622 6000 FC78        bra     check_ov_error  ; Will enable ints
1820
```



```
1823
1824
1825                    *       ****    *   *   ****    *   *   *****   *****   *****   ****
1826                    *       *   *   *   *   *   *   *   *   *       *       *       *   *
1827                    *       *   *    * *    *   *   *   *   *       *       *       *   *
1828                    *       ****      *     ****    *   *   *****   *****   *****   ****
1829                    *       *  *      * *    *   *   *   *   *       *       *       *  *
1830                    *       *   *   *   *   *   *   *   *   *       *       *       *   *
1831                    *       *   *   *   *   ****     ***    *       *       *****   *   *
1832
1833
1834
1835
1836
1837                    ********************************************************************************
1838                    *                                                                              *
1839                    *       routine set_RXBUF_a6:   Routine to set a6 as the base address          *
1840                    *                               pointer to whichever Rx buffer is being used.  *
1841                    *                                                                              *
1842                    *       At entry:                                                              *
1843                    *               a2.l =  data buffer base address (shifted, +1+selectcode)      *
1844                    *               a3.l =  card base address ($00xx0001)                          *
1845                    *               a4.l =  pointer to sc_subtabletype structure                   *
1846                    *                                                                              *
1847                    *       Upon exit:                                                             *
1848                    *               a6.l =  Base address of DATA_BUFFERS[WHICH_RXBUF]              *
1849                    *               This also bashes d0.                                           *
1850                    *                                                                              *
1851                    ********************************************************************************
1852
1853           0000 0626 set_RXBUF_a6 equ *
1854    00000626 4280             clr.l   d0                      ; Setup d0.l=offset
1855    00000628 102C 0037        move.b  which_RXbuf(a4),d0      ; to which Rx buffer
1856    0000062C E980             asl.l   #4,d0                   ; being used
1857    0000062E 4DF2 0010        lea     DATA_AREA(a2,d0),a6
1858    00000632 4E75             rts
```

```
1860                    ********************************************************************************
1861                    *                                                                              *
1862                    *       routine RX_BUFF_bytes:  Function which returns the number of           *
1863                    *               =============   characters until the first control block in     *
1864                    *                               the Receive Buffer.  If there are no control   *
1865                    *                               blocks, this just returns the number of        *
1866                    *                               characters in the buffer.  This only works     *
1867                    *                               on the current Rx data buffer, and does not    *
1868                    *                               extract TERM=255 control blocks!               *
1869                    *                                                                              *
1870                    *       At entry:                                                              *
1871                    *               a2.l =  data buffer base address (shifted, +1+selectcode)      *
1872                    *               a3.l =  card base address ($00xx0001)                          *
1873                    *               a4.l =  pointer to sc_subtabletype structure                   *
1874                    *                                                                              *
1875                    *       Upon exit:                                                             *
1876                    *               d0.l =  Number of characters.                                  *
1877                    *               a1, d4 and d5 are left with values from find_RX_DATA.          *
1878                    *               This also bashes a0, d1 and d2.                                *
1879                    *                                                                              *
1880                    *       This routine uses the card's SEMAPHORE to gain access.                 *
1881                    *                                                                              *
1882                    *       This routine calls gain_access, release_access, and find_RX_DATA.      *
1883                    *                                                                              *
1884                    ********************************************************************************
1885
1886            0000 0634 RX_BUFF_bytes equ *
1887    00000634 6100 FA5E      bsr     find_RX_DATA            ; Setup a1 = data buffer base addr
1888                    *                                                       d4 = end of data buffer addr
1889                    *                                                       d5 = RXDATABUFF_SIZE
1890    00000638 2F0E           move.l  a6,-(sp)
1891    0000063A 61EA           bsr     set_RXBUF_a6
1892
1893    0000063C 4280           clr.l   d0                      ; Get garbage out of top of d0, d1
1894    0000063E 4281           clr.l   d1
1895    00000640 030A 000C      movep.w CTRL_AREA+EMPTY(a2),d1  ; Fetch pointers (bytes in wrong order)
1896    00000644 6100 F9E4      bsr     gain_access             ; Need access to FILL pointers
1897    00000648 010E 0008      movep.w FILL(a6),d0
1898    0000064C 050A 0008      movep.w CTRL_AREA+FILL(a2),d2
1899    00000650 6100 F9FA      bsr     release_access
1900    00000654 B242           cmp.w   d2,d1                   ; If the two ctrl block pointers are not
1901    00000656 670A           beq.s   RBb1                    ; equal, then we want to use the pointer
1902    00000658 EE59           ror.w   #7,d1                   ; field from the next control block to
1903    0000065A D28B           add.l   a3,d1                   ; indicate how much data may be removed
1904    0000065C 2041           movea.l d1,a0
1905    0000065E 0108 0000      movep.w POINTER(a0),d0          ; --- Use it as the "FILL" pointer
1906
1907    00000662 E058   RBb1    ror.w   #8,d0                   ; Switch bytes for FILL
1908    00000664 030E 000C      movep.w EMPTY(a6),d1            ; and get EMPTY and switch bytes
1909    00000668 E059           ror.w   #8,d1                   ; d0="FILL", d1=EMPTY
1910
1911    0000066A 9041           sub.w   d1,d0                   ; Compute d0 := FILL-EMPTY
1912    0000066C 6C02           bge.s   RBb2
1913    0000066E D045           add.w   d5,d0                   ; If negative, add data buffer size
1914    00000670 2C5F   RBb2    movea.l (sp)+,a6
1915    00000672 4E75           rts     ; Now d0 = ("FILL"-EMPTY) mod SIZE --- of data buffer
```



```
1917                    ********************************************************************************
1918                    *                                                                              *
1919                    *       routine getctrlblk:     Routine which gets a control block from the    *
1920                    *               ==========      Receive buffer.  It must have already been     *
1921                    *                               determined that there is a control block at    *
1922                    *                               the front of the buffer, since this routine    *
1923                    *                               does NOT check for that condition.  The TERM   *
1924                    *                               and MODE fields of the removed block are left  *
1925                    *                               in the appropriate (.term and .mode) in the    *
1926                    *                               sc_subtabletype structure.                     *
1927                    *                                                                              *
1928                    *       At entry:                                                              *
1929                    *               a2.l =  RX buffer record base address from find_RXBUF          *
1930                    *               a3.l =  card base address ($00xx0001)                          *
1931                    *               a4.l =  pointer to sc_subtabletype structure                   *
1932                    *                                                                              *
1933                    *       Upon exit:                                                             *
1934                    *               sc_subtabletype.term =  TERM field of control block (8 bits)   *
1935                    *               sc_subtabletype.mode =  MODE field of control block (8 bits)   *
1936                    *               a1, d4 and d5 are left with the values from find_CTRL_AREA.    *
1937                    *               This bashes d0, d2, and a0.                                    *
1938                    *                                                                              *
1939                    *       This routine uses the card's SEMAPHORE to gain access.                 *
1940                    *                                                                              *
1941                    *       This routine calls gain_access, release_access, and find_CTRL_AREA.    *
1942                    *                                                                              *
1943                    ********************************************************************************
1944
1945            0000 0674 getctrlblk equ  *
1946    00000674 6100 FA38      bsr     find_CTRL_AREA          ; Setup a1 = ctrl buffer base addr
1947                    *                                                       d4 = end of ctrl buffer addr
1948                    *                                                       d5 = TRCTRLBUFF_SIZE
1949
1950    00000678 4280           clr.l   d0                      ; Clear top of d0
1951    0000067A 010A 000C      movep.w CTRL_AREA+EMPTY(a2),d0  ; Get control buffer EMPTY pointer
1952    0000067E EE58           ror.w   #7,d0                   ; Now make it into a 68000 pointer
1953    00000680 D08B           add.l   a3,d0
1954    00000682 2040           movea.l d0,a0                   ; Move to a0 so we can use it
1955
1956    00000684 1968 0004      move.b  TERMFIELD(a0),term(a4)  ; Store term & mode fields
                 003C
1957    0000068A 1968 0006      move.b  MODEFIELD(a0),mode(a4)
                 003D
1958
1959    00000690 D07C 0008      add.w   #CTRLBLKSIZE,d0         ; Bump pointer by control block size
1960    00000694 B840           cmp.w   d0,d4                   ; and check for wraparound.
1961    00000696 6602           bne.s   gcb1
1962    00000698 3009           move.w  a1,d0                   ; If so, set to front of buffer
1963    0000069A 0880 0000 gcb1 bclr    #0,d0                   ; Make it into a Z80
1964    0000069E EF58           rol.w   #7,d0                   ; type pointer with bytes reversed
1965    000006A0 6100 F988      bsr     gain_access             ; Now store the updated EMPTY pointer
1966    000006A4 018A 000C      movep.w d0,CTRL_AREA+EMPTY(a2)
1967    000006A8 6100 F9A2      bsr     release_access
1968    000006AC 4E75           rts     ;<<<CAN'T COMBINE WITH ABOVE!!!!!
```

```
1970                    ********************************************************************************
1971                    *                                                                              *
1972                    *       routine getchars:       Routine which takes characters from the        *
1973                    *               ========        Receive buffer and puts them in the area       *
1974                    *                               pointed to by sc_subtabletype.data_address     *
1975                    *                               and sized by sc_subtabletype.data_number.      *
1976                    *                               The number of characters                       *
1977                    *                               actually transfered is the minimum of:         *
1978                    *                               (1) the number of characters available before  *
1979                    *                               the first Receive buffer control block;        *
1980                    *                               (2) sc_subtabletype.data_number;               *
1981                    *                               and (3) the number of characters               *
1982                    *                               available until the Receive buffer wraparound  *
1983                    *                               point.  THIS NUMBER MAY BE ZERO!               *
1984                    *                               This alters data_address and data_number to    *
1985                    *                               reflect where to start going next time this    *
1986                    *                               is called.  The criteria for ending the        *
1987                    *                               transfer at a higher level must be determined  *
1988                    *                               by data_number, RX_stuff_avail and             *
1989                    *                               ctrl_blk_next/getctrlblk.                      *
1990                    *                                                                              *
1991                    *       At entry:                                                              *
1992                    *               a2.l =  RX buffer record base address from find_RXBUF          *
1993                    *               a3.l =  card base address ($00xx0001)                          *
1994                    *               a4.l =  pointer to sc_subtabletype structure                   *
1995                    *                                                                              *
1996                    *       Upon exit:                                                             *
1997                    *               data_address and data_number are updated, plus the EMPTY       *
1998                    *               pointer in the card's Receive data buffer.                     *
1999                    *               In sc_subtabletype last_enter_term and last_enter_mode are     *
2000                    *               zeroed if any data is moved.                                   *
2001                    *               a1 and d4 are left with the values from find_RX_DATA.          *
2002                    *               This bashes d0, d1, d2, d3, d4, d5, a0, and a1.                *
2003                    *                                                                              *
2004                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2005                    *                                                                              *
2006                    *       This routine calls gain_access, release_access, and RX_BUFF_bytes.     *
2007                    *                                                                              *
2008                    ********************************************************************************
2009
2010            0000 06AE getchars equ     *
2011    000006AE 6184               bsr     RX_BUFF_bytes           ; Setup a1 = data buffer base addr
2012                    *                                                       d3 = offset to which Rxbuff used
2013                    *                                                       d4 = end of data buffer addr
2014                    *                                                       d5 = RXDATABUFF_SIZE
2015    000006B0 2600               move.l  d0,d3                   ; d3.l = available characters
2016
2017    000006B2 48E7 0022          movem.l a2/a6,-(sp)             ; Saved for local use
2018    000006B6 6100 FF6E          bsr     set_RXBUF_a6
2019
2020    000006BA 4280               clr.l   d0
2021    000006BC 010E 000C          movep.w EMPTY(a6),d0            ; Get RXDATABUFF_EMPTY and make
2022    000006C0 EE58               ror.w   #7,d0                   ; it into a 68000 pointer
2023    000006C2 D08B               add.l   a3,d0
2024    000006C4 2440               movea.l d0,a2                   ; Save EMPTY for later!
2025
2026    000006C6 9084               sub.l   d4,d0
```



```
2027    000006C8 4480               neg.l   d0                      ; d0 = wraparound address - EMPTY
2028    000006CA C0BC 0000          and.l   #$0000FFFE,d0
                 FFFE
2029    000006D0 E258               ror.w   #1,d0                   ; d0.l = number of bytes till wraparound
2030
2031    000006D2 B680               cmp.l   d0,d3                   ; If d0>d3 then set d0 := d3
2032    000006D4 6E02               bgt.s   gc1
2033    000006D6 2003               move.l  d3,d0
2034
2035    000006D8 242C 0042 gc1      move.l  data_number(a4),d2      ; Fetch number of positions available
2036    000006DC B480               cmp.l   d0,d2                   ; If d0>d2 then set d0 := d2
2037    000006DE 6E02               bgt.s   gc2
2038    000006E0 2002               move.l  d2,d0
2039
2040    000006E2 2600       gc2     move.l  d0,d3                   ; d3.l saves number of chars actually
2041                    *                                           ; transferred below
2042    000006E4 6736               beq.s   gcdone                  ; If zero, no work to be done
2043    000006E6 426C 0038          clr.w   last_enter_term(a4)     ; This also clears last_enter_mode.
2044    000006EA 5340               subq.w  #1,d0                   ; Make offset correct for dbf instr.
2045    000006EC 206C 003E          movea.l data_address(a4),a0     ; Get character pointer into a0
2046
2047    000006F0 10D2       gcloop  move.b  (a2),(a0)+              ; Transfer a character & bump dest ptr
2048    000006F2 544A               addq.w  #2,a2                   ; Bump source pointer (odd bytes)
2049    000006F4 51C8 FFFA          dbf     d0,gcloop               ; Then decrement d0 & loop
2050
2051    000006F8 9483               sub.l   d3,d2                   ; Decrement datacnt by # bytes
2052    000006FA 2942 0042          move.l  d2,data_number(a4)      ; Now store adjusted address and
2053    000006FE 2948 003E          move.l  a0,data_address(a4)     ; number fields
2054
2055    00000702 220A               move.l  a2,d1                   ; Store pointer for computations
2056    00000704 B88A               cmp.l   a2,d4                   ; Now check to see if EMPTY was moved
2057    00000706 6602               bne.s   gc3                     ; past end of buffer.  If so, set to
2058    00000708 2209               move.l  a1,d1                   ; the front of the buffer.
2059    0000070A 0881 0000 gc3      bclr    #0,d1                   ; Fix up the 68000 pointer to be the
2060    0000070E EF59               rol.w   #7,d1                   ; card's type of pointer
2061    00000710 6100 F918          bsr     gain_access
2062    00000714 038E 000C          movep.w d1,EMPTY(a6)            ; Remember d1 = card's EMPTY pointer.
2063    00000718 6100 F932          bsr     release_access
2064    0000071C 4CDF 4400 gcdone   movem.l (sp)+,a6/a2
2065    00000720 4E75               rts
```

```
2067                    ********************************************************************************
2068                    *                                                                              *
2069                    *       routine RX_stuff_avail: Routine which determines whether there is      *
2070                    *               ==============  ANYTHING (data or control blocks) in the       *
2071                    *                               Receive buffer.  This consumes any TERM=255    *
2072                    *                               control blocks before returning the function.  *
2073                    *                                                                              *
2074                    *       At entry:                                                              *
2075                    *               a2.l =  RX buffer record base address from find_RXBUF          *
2076                    *               a3.l =  card base address ($00xx0001)                          *
2077                    *               a4.l =  pointer to sc_subtabletype structure                   *
2078                    *                                                                              *
2079                    *       Upon exit:                                                             *
2080                    *               d0.l =  $00 if buffer is empty,                                *
2081                    *                       $01 if ctrl buffer is empty and data buffer is not,    *
2082                    *                       $02 if data buffer is empty and ctrl buffer is not,    *
2083                    *                       $03 if both data and ctrl buffers are not empty.       *
2084                    *               a1 and d4 are left with the values from find_RX_DATA.          *
2085                    *               This bashes d0, d1, d2, d3, d4, d5, a0 and a1.                 *
2086                    *                                                                              *
2087                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2088                    *                                                                              *
2089                    *       This routine calls gain_access and release_access.                     *
2090                    *                                                                              *
2091                    ********************************************************************************
2092
2093          0000 0722 RX_stuff_avail equ *
2094    00000722 6100 F970            bsr     find_RX_DATA            ; Setup a1 = data buffer base addr
2095                    *                                                       d4 = end of data buffer addr
2096                    *                                                       d5 = RXDATABUFF_SIZE
2097    00000726 2F0E                 move.l  a6,-(sp)
2098    00000728 6100 FEFC            bsr     set_RXBUF_a6
2099
2100    0000072C 6100 F8FC            bsr     gain_access
2101    00000730 070E 0008            movep.w FILL(a6),d3             ; Fetch FILL & EMPTY (bytes reversed but
2102    00000734 030A 0008            movep.w CTRL_AREA+FILL(a2),d1
2103    00000738 6100 F912            bsr     release_access          ; we're just checking equality)
2104    0000073C 4282                 clr.l   d2
2105    0000073E 4280                 clr.l   d0
2106    00000740 050A 000C            movep.w CTRL_AREA+EMPTY(a2),d2
2107    00000744 B441                 cmp.w   d1,d2                   ; Compare ctrl buff FILL & EMPTY
2108    00000746 660E                 bne.s   setbit1                 ; If not equal, then set bit 1
2109    00000748 050E 000C chkdata    movep.w EMPTY(a6),d2
2110    0000074C B443                 cmp.w   d3,d2                   ; Compare data buff FILL & EMPTY
2111    0000074E 6702                 beq.s   return
2112    00000750 5200                 addq.b  #1,d0                   ; And set bit 0 if not equal
2113    00000752 2C5F       return    movea.l (sp)+,a6
2114    00000754 4E75                 rts
2115
2116    00000756 5400       setbit1   addq.b  #2,d0                   ; Set "ctrl not empty" bit
2117    00000758 EE5A                 ror.w   #7,d2                   ; Something in control buffer - see if
2118                    *                                             ; this control block is at the head of
2119    0000075A D48B                 add.l   a3,d2                   ; the queue (bytes reversed!)
2120    0000075C 2042                 movea.l d2,a0
2121    0000075E 0308 0000            movep.w POINTER(a0),d1
2122    00000762 050E 000C            movep.w EMPTY(a6),d2
2123    00000766 B242                 cmp.w   d2,d1                   ; if POINTER field<>DATABUFF_EMPTY
```



```
2124    00000768 66DE                 bne.s   chkdata                 ;   then go check data buff
2125    0000076A 0C28 00FF            cmpi.b  #255,TERMFIELD(a0)      ; else if it's a TERM=255 control block
                 0004
2126    00000770 66D6                 bne.s   chkdata                 ; No, go back and check data buff
2127
2128    00000772 6100 FF00            bsr     getctrlblk              ; Otherwise consume the control block
2129    00000776 196C 003D            move.b  mode(a4),which_RXbuf(a4) ;and switch to new data buffer
                 0037
2130    0000077C 2C5F                 movea.l (sp)+,a6
2131    0000077E 60A2                 bra.s   RX_stuff_avail          ; And go back and re-compute result
```

```
2133                        ********************************************************************************
2134                        *                                                                              *
2135                        *       routine ctrlblknext:    Routine which determines whether the next      *
2136                        *               ===========     thing to be consumed from the Receive buffer   *
2137                        *                               is a control block.  THE RESULT OF THIS       *
2138                        *                               FUNCTION IS NOT VALID UNLESS RX_BUFFER_empty   *
2139                        *                               RETURNS FALSE!!!                               *
2140                        *                                                                              *
2141                        *       At entry:                                                              *
2142                        *               a2.l =  RX buffer record base address from find_RXBUF          *
2143                        *               a3.l =  card base address ($00xx0001)                          *
2144                        *                                                                              *
2145                        *       Upon exit:                                                             *
2146                        *               d0.b =  $FF if control block is next, $00 if data is next.     *
2147                        *               This bashes d2, d5 and a0.                                     *
2148                        *                                                                              *
2149                        ********************************************************************************
2150
2151             0000 0780 ctrlblknext equ *
2152    00000780 6100 F8A8         bsr     gain_access
2153    00000784 050A 0008         movep.w CTRL_AREA+FILL(a2),d2   ; Check if ctrl buffer is empty
2154    00000788 6100 F8C2         bsr     release_access
2155    0000078C 4280              clr.l   d0
2156    0000078E 010A 000C         movep.w CTRL_AREA+EMPTY(a2),d0  ; Fetch ctrl buffer EMPTY pointer
2157    00000792 B440              cmp.w   d0,d2                   ; If equal then return d0.b=$00
2158    00000794 6728              beq.s   cbn1
2159    00000796 EE58              ror.w   #7,d0
2160    00000798 D08B              add.l   a3,d0
2161    0000079A 2040              movea.l d0,a0
2162    0000079C 0108 0000         movep.w POINTER(a0),d0          ; Fetch the POINTER field from the
2163
2164    000007A0 4285              clr.l   d5                      ; Setup d5.l=offset
2165    000007A2 1A2C 0037         move.b  which_RXbuf(a4),d5      ; to which Rx buffer
2166    000007A6 E985              asl.l   #4,d5                   ; being used
2167
2168    000007A8 48E7 8002         movem.l d0/a6,-(sp)
2169    000007AC 6100 FE78         bsr     set_RXBUF_a6
2170    000007B0 050E 000C         movep.w EMPTY(a6),d2            ; first ctrl block and compare to the
2171    000007B4 4CDF 4001         movem.l (sp)+,d0/a6
2172
2173    000007B8 B440              cmp.w   d0,d2                   ; data buffer EMPTY pointer
2174    000007BA 57C0              seq     d0                      ; Then set d0 if equal
2175    000007BC 4E75              rts
2176
2177    000007BE 4280       cbn1   clr.l   d0
2178    000007C0 4E75              rts
2179
```



```
2182
2183
2184                        *       *****   ****     ***    *   *    ***    *****   *****   ****
2185                        *         *     *   *   *   *   ** *   *   *   *       *       *   *
2186                        *         *     *   *   *   *   * * *   *       *       *       *   *
2187                        *         *     ****    *****   *  **    ***    ***     ***     ****
2188                        *         *     *  *    *   *   *   *       *   *       *       *  *
2189                        *         *     *   *   *   *   *   *   *   *   *       *       *   *
2190                        *         *     *   *   *   *   *   *    ***    *       *****   *   *
2191
2192
2193
2194
2195             0000 0006 in_timeout      equ     6  [UNCALIBRATED 1 MS]
2196             0000 0007 out_timeout     equ     7  [UNCALIBRATED 1 MS]
2197
2198
2199                        ************************************************
2200                        *                                              *
2201                        *   procedure OUTPUT_DATA (                    *
2202                        *               var SCT:  select_code_table;   *
2203                        *               PTR:      ^ data_bytes;        *
2204                        *               COUNT:    longword       );    *
2205                        *                                              *
2206                        *   This operation may hang waiting for space. *
2207                        *                                              *
2208                        *   This routine calls find_TXBUF and putchars. *
2209                        *                                              *
2210                        ************************************************
2211
2212             0000 07C2 output_data equ *
2213    000007C2 205F              movea.l (sp)+,a0
2214    000007C4 221F              move.l  (sp)+,d1        ; COUNT
2215    000007C6 225F              movea.l (sp)+,a1        ; PTR
2216    000007C8 285F              movea.l (sp)+,a4        ; SCT
2217    000007CA 4850              pea     (a0)
2218    000007CC 266C 0020         movea.l c_adr(a4),a3
2219    000007D0 528B              addq.l  #1,a3
2220
2221    000007D2 2949 003E         move.l  a1,data_address(a4)     ; initialize address/count
2222    000007D6 2941 0042         move.l  d1,data_number(a4)
2223
2224    000007DA 6100 FAC0         bsr     check_ov_error          ; Escape if o/v error
2225    000007DE 6100 FDB6         bsr     wait_outxfrdone
2226
2227    000007E2 296C 002E         move.l  timeout(a4),timeout_counter(a4)
                004A
2228    000007E8 297C 0000         move.l  #out_timeout,inner_counter(a4)
                0007 004E
2229
2230    000007F0 6100 F878         bsr     find_TXBUF              ; Set up a2.l = buffer descriptor record
2231                        *                                                       base address
2232    000007F4 6728              beq.s   outdone
2233
2234    000007F6 4AAC 0042 out_2   tst.l   data_number(a4)         ; And transfer characters until done
2235    000007FA 6722              beq.s   outdone
2236    000007FC 6100 019C         bsr     putchars
```

```
00000800 53AC 004E          subq.l  #1,inner_counter(a4)    ; Test for timeout condition
00000804 66F0               bne.s   out_2
00000806 297C 0000          move.l  #out_timeout,inner_counter(a4)
         0007 004E
0000080E 4AAC 004A          tst.l   timeout_counter(a4)
00000812 67E2               beq.s   out_2
00000814 53AC 004A          subq.l  #1,timeout_counter(a4)
00000818 6700 FAA2          beq     time_err                ; if so, escape
0000081C 60D8               bra.s   out_2

0000081E 4E75       outdone rts
```



```
                    ************************************************
                    *                                              *
                    *   procedure ENTER_DATA (                     *
                    *               var SCT:  select_code_table;   *
                    *               PTR:        ^ data_bytes;      *
                    *               var COUNT:longword       );    *
                    *                                              *
                    *  COUNT initially passes the number of bytes  *
                    *  which the upper level wants to read.  THE   *
                    *  ROUTINE DOES NOT NECESSARILY READ THIS MANY! *
                    *  Upon exit COUNT will be reflect the number  *
                    *  of data bytes entered, whether or not there *
                    *  is an escape.                               *
                    *                                              *
                    *  escape(EOD): Terminated by reaching a control*
                    *     block.  TERM&MODE may be read with STATUS *
                    *     9 and 10.                                *
                    *                                              *
                    *       This routine calls find_RXBUF,         *
                    *       getctrlblk, getchars, ctrlblknext, and *
                    *       RX_BUFFER_EMPTY.                       *
                    *                                              *
                    ************************************************

          0000 0820 enter_data equ  *
00000820 205F               movea.l (sp)+,a0
00000822 245F               movea.l (sp)+,a2        ; addr(COUNT)
00000824 225F               movea.l (sp)+,a1        ; PTR
00000826 285F               movea.l (sp)+,a4        ; SCT
00000828 4850               pea     (a0)
0000082A 266C 0020          movea.l c_adr(a4),a3
0000082E 528B               addq.l  #1,a3

00000830 2949 003E          move.l  a1,data_address(a4)     ; initialize address

00000834 6100 FA66          bsr     check_ov_error          ; Escape if o/v error
00000838 6100 FD66          bsr     wait_inxfrdone

0000083C 296C 002E          move.l  timeout(a4),timeout_counter(a4)
         004A
00000842 297C 0000          move.l  #in_timeout,inner_counter(a4)
         0006 004E

0000084A 2952 0042          move.l  (a2),data_number(a4)
0000084E 2F0A               move.l  a2,-(sp)
00000850 600A               bra.s   in_1

00000852 6100 F8B8 in_0     bsr     eir
00000856 4AAC 0042          tst.l   data_number(a4)         ; See if all characters transferred
0000085A 675A               beq.s   in_exit                 ; If so, leave

0000085C 6100 F810 in_1     bsr     find_RXBUF              ; Set up a2.l = buffer descriptor record
                            *                                             base address
00000860 53AC 004E          subq.l  #1,inner_counter(a4)    ; Test for timeout condition
00000864 6622               bne.s   in_4
00000866 297C 0000          move.l  #in_timeout,inner_counter(a4)
         0006 004E
```

```
2302    0000086E 4AAC 004A          tst.l   timeout_counter(a4)
2303    00000872 6714               beq.s   in_4
2304    00000874 53AC 004A          subq.l  #1,timeout_counter(a4)
2305    00000878 660E               bne.s   in_4
2306    0000087A 205F               movea.l (sp)+,a0
2307    0000087C 2010               move.l  (a0),d0
2308    0000087E 90AC 0042          sub.l   data_number(a4),d0
2309    00000882 2080               move.l  d0,(a0)
2310    00000884 6000 FA36          bra     time_err
2311
2312    00000888 6100 F88A in_4     bsr     dir
2313    0000088C 6100 FE94          bsr     RX_stuff_avail          ; See if buffer is empty
2314    00000890 4A00               tst.b   d0                      ; If so, just sit here & wait
2315    00000892 67BE               beq.s   in_0
2316
2317    00000894 6100 FEEA          bsr     ctrlblknext             ; If a control block is next, then
2318    00000898 4A00               tst.b   d0
2319    0000089A 6736               beq.s   in_3
2320
2321    0000089C 6100 FDD6          bsr     getctrlblk              ; Get it, and check for the special
2322    000008A0 0C2C 00FF          cmpi.b  #255,term(a4)           ; case TERM=255
                 003C
2323    000008A6 6608               bne.s   in_2
2324    000008A8 196C 003D          move.b  mode(a4),which_RXbuf(a4) ;If so, do the buffer switch
                 0037
2325    000008AE 60AC               bra.s   in_1                    ; And go back for more
2326
2327    000008B0 396C 003C in_2     move.w  term_and_mode(a4),last_enter_term(a4)
                 0038
2328                        *                                       ; Otherwise save the control block & leave
2329    000008B6 6100 F854 in_exit  bsr     eir
2330    000008BA 205F               movea.l (sp)+,a0
2331    000008BC 2010               move.l  (a0),d0
2332    000008BE 90AC 0042          sub.l   data_number(a4),d0
2333    000008C2 2080               move.l  d0,(a0)
2334    000008C4 4AAC 0042          tst.l   data_number(a4)
2335    000008C8 6700 FF54          beq     outdone                 ; If nonzero then early EOI; escape
2336    000008CC 7016               moveq   #EOD_SEEN,d0
2337    000008CE 6000 F9EE          bra     escape
2338
2339    000008D2 4AAC 0042 in_3     tst.l   data_number(a4)         ; see if chars to transfer
2340    000008D6 67DE               beq.s   in_exit                 ; yes, go do it
2341
2342    000008D8 6100 FDD4          bsr     getchars                ; move some data
2343    000008DC 6000 FF74          bra     in_0                    ; & go back to check for ctrl blk
```



```
2345                        ************************************************
2346                        *                                              *
2347                        *   procedure OUTPUT_END (                     *
2348                        *               var SCT:  select_code_table );  *
2349                        *                                              *
2350                        *   Equivalent to the BASIC OUTPUT Sc;END.     *
2351                        *                                              *
2352                        *   This operation may hang waiting for space.  *
2353                        *                                              *
2354                        *   This routine calls find_TXBUF and          *
2355                        *   try_sending_EOF.                           *
2356                        *                                              *
2357                        ************************************************
2358
2359             0000 08E0 output_end equ *
2360    000008E0 205F               movea.l (sp)+,a0
2361    000008E2 285F               movea.l (sp)+,a4        ; SCT
2362    000008E4 4850               pea     (a0)
2363    000008E6 266C 0020          movea.l c_adr(a4),a3
2364    000008EA 528B               addq.l  #1,a3
2365
2366    000008EC 6100 F9AE          bsr     check_ov_error          ; Escape if o/v error
2367    000008F0 6100 FCA4          bsr     wait_outxfrdone
2368
2369    000008F4 6100 F774          bsr     find_TXBUF              ; Set up a2.l = buffer descriptor record
2370                        *                                               base address
2371    000008F8 296C 002E          move.l  timeout(a4),outer_tx_count(a4)
                 0046
2372    000008FE 297C 0000          move.l  #sEtimeout,inner_tx_count(a4)
                 000B 0052
2373
2374    00000906 6100 0180 trysend  bsr     try_sending_EOF
2375    0000090A 4A00               tst.b   d0
2376    0000090C 661E               bne.s   sentEOF
2377
2378    0000090E 53AC 0052          subq.l  #1,inner_tx_count(a4)
2379    00000912 66F2               bne.s   trysend
2380    00000914 297C 0000          move.l  #sEtimeout,inner_tx_count(a4)
                 000B 0052
2381    0000091C 4AAC 0046          tst.l   outer_tx_count(a4)
2382    00000920 67E4               beq.s   trysend
2383    00000922 53AC 0046          subq.l  #1,outer_tx_count(a4)
2384    00000926 66DE               bne.s   trysend
2385    00000928 6000 F992          bra     time_err
2386
2387    0000092C 4E75       sentEOF rts
```

```
2389                    ************************************************
2390                    *                                              *
2391                    *   procedure CONTROL_BFD (                    *
2392                    *               var SCT:  select_code_table;   *
2393                    *               REG:      0..127;              *
2394                    *               VAL:      0..255);             *
2395                    *                                              *
2396                    *   Control register 0 is intercepted and      *
2397                    *   if MODE=0 no action is performed, otherwise *
2398                    *   the card is reset IMMEDIATELY.             *
2399                    *                                              *
2400                    *   This operation may hang waiting for space. *
2401                    *                                              *
2402                    *   The ranges of REG & VAL are not checked    *
2403                    *   for validity.                              *
2404                    *                                              *
2405                    *   This routine calls find_TXBUF and putctrlblk*
2406                    *                                              *
2407                    ************************************************
2408
2409          0000 092E control_bfd equ *
2410    0000092E 205F               movea.l (sp)+,a0
2411    00000930 321F               move.w  (sp)+,d1        ; VAL
2412    00000932 341F               move.w  (sp)+,d2        ; REG
2413    00000934 285F               movea.l (sp)+,a4        ; SCT
2414    00000936 4850               pea     (a0)
2415    00000938 266C 0020          movea.l c_adr(a4),a3
2416    0000093C 528B               addq.l  #1,a3
2417
2418    0000093E 6100 F95C          bsr     check_ov_error  ; Escape if o/v error
2419                    *                                   (tm) moved  12/02/81
2420    00000942 1942 003C          move.b  d2,term(a4)     (tm) ; Intercept CONTROL 0
2421    00000946 6710               beq.s   ctrl0           (tm)
2422
2423    00000948 6100 FC4C          bsr     wait_outxfrdone
2424
2425    0000094C 1941 003D          move.b  d1,mode(a4)
2426
2427    00000950 6100 F718          bsr     find_TXBUF              ; Set up a2.l = buffer descriptor record
2428    00000954 6000 00B4          bra     putctrlblk              ;               base address
2429
2430    00000958 4A01       ctrl0   tst.b   d1
2431    0000095A 6708               beq.s   ctrldun
2432    0000095C 6100 FC42          bsr     wait_inxfrdone
2433    00000960 6000 F91A          bra     do_reset
2434    00000964 4E75       ctrldun rts
2435
```



```
2438
2439
2440                    *       *****   *   *   ****    *   *   *****   *****   *****   ****
2441                    *         *     *   *   *   *   *   *   *       *       *       *   *
2442                    *         *      * *    *   *   *   *   *       *       *       *   *
2443                    *         *       *     ****    *   *   *****   *****   *****   ****
2444                    *         *      * *    *   *   *   *   *       *       *       *  *
2445                    *         *     *   *   *   *   *   *   *       *       *       *   *
2446                    *         *     *   *   ****     ***    *       *       *****   *   *
2447
2448
2449
2450          0000 000B sEtimeout       equ     11    [UNCALIBRATED]
2451          0000 000B pcbtimeout      equ     11    [CALIBRATED 1 MS]
2452
2453
2454                    ********************************************************************************
2455                    *                                                                              *
2456                    *       routine TXCTRLBUFFroom: Function which returns the number of byte      *
2457                    *               ==============  positions as yet unused in the Transmit ctrl   *
2458                    *                               Buffer.                                        *
2459                    *                                                                              *
2460                    *       routine TXDATABUFFroom: Function which returns the number of byte      *
2461                    *               ==============  positions as yet unused in the Transmit data   *
2462                    *                               Buffer.                                        *
2463                    *                                                                              *
2464                    *       At entry:                                                              *
2465                    *               a2.l =  TXBUFF base address (shifted, +1+selectcode)           *
2466                    *               a3.l =  card base address ($00xx0001)                          *
2467                    *               d5.l =  TXDATABUFF_SIZE or TXCTRLBUFF_SIZE                     *
2468                    *                       (unshifted, not adjusted)                              *
2469                    *                                                                              *
2470                    *       Upon exit:                                                             *
2471                    *               d0.l =  TXDATABUFF_FILL or TXCTRLBUFF_FILL (unshifted)         *
2472                    *               d3.l =  Number of bytes left                                   *
2473                    *               This also bashes d2.                                           *
2474                    *                                                                              *
2475                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2476                    *                                                                              *
2477                    *       This routine calls gain_access and release_access.                     *
2478                    *                                                                              *
2479                    ********************************************************************************
2480
2481          0000 0966 TXCTRLBUFFroom equ *
2482    00000966 4280               clr.l   d0                      ; Get garbage out of top of d0&d3
2483    00000968 4283               clr.l   d3
2484    0000096A 010A 0008          movep.w CTRL_AREA+FILL(a2),d0
2485    0000096E 6100 F6BA          bsr     gain_access             ; Need access to EMPTY
2486    00000972 070A 000C          movep.w CTRL_AREA+EMPTY(a2),d3  ; Fetch pointers (bytes in wrong order)
2487    00000976 6010               bra.s   room1
2488
2489          0000 0978 TXDATABUFFroom equ *
2490    00000978 4280               clr.l   d0                      ; Get garbage out of top of d0&d3
2491    0000097A 4283               clr.l   d3
2492    0000097C 010A 0018          movep.w DATA_AREA+FILL(a2),d0
2493    00000980 6100 F6A8          bsr     gain_access             ; Need access to EMPTY
2494    00000984 070A 001C          movep.w DATA_AREA+EMPTY(a2),d3  ; Fetch pointers (bytes in wrong order)
```

```
00000988 6100 F6C2 room1   bsr     release_access
                                                
0000098C E058              ror.w   #8,d0                   ; Switch bytes in d0 & d3
0000098E E05B              ror.w   #8,d3
                                                
00000990 9640              sub.w   d0,d3                   ; Compute d3 := EMPTY-FILL
00000992 5343              subq.w  #1,d3                   ; (EMPTY-FILL-1)
00000994 6C02              bge.s   room2
00000996 D645              add.w   d5,d3                   ; If negative, add size
                                                
00000998 4E75      room2   rts                             ; Return (EMPTY-FILL-1) mod SIZE
```



```
                           ********************************************************************************
                           *                                                                              *
                           *       routine putchars:       Routine which takes characters from the        *
                           *               ========        area sc_subtabletype.data_address sized by     *
                           *                               sc_subtabletype.data_number and moves          *
                           *                               them to the Transmit buffer.  The number of    *
                           *                               characters actually transfered is the minimum  *
                           *                               of: (1) the number of characters available;    *
                           *                               (2) the number of byte positions left in the   *
                           *                               Transmit buffer; and (3) the number of byte    *
                           *                               positions in the Transmit buffer until the     *
                           *                               wraparound point.  THIS NUMBER CAN BE ZERO.    *
                           *                               This alters data_address and data_number to    *
                           *                               reflect where to start going next time this    *
                           *                               is called.  The entire transfer is done when   *
                           *                               data_number goes to zero.                      *
                           *                                                                              *
                           *       At entry:                                                              *
                           *               a2.l =  TX buffer record base address (shifted, +1+selectcode) *
                           *               a3.l =  card base address ($00xx0001)                          *
                           *               a4.l =  pointer to sc_subtabletype structure                   *
                           *                                                                              *
                           *       Upon exit:                                                             *
                           *               data_number and data_address are updated, plus FILL in the     *
                           *               card's Transmit buffer.                                        *
                           *               a1, d4 and d5 are left with the values from find_DATA_AREA.    *
                           *               This bashes d0, d1, d2, d3, d4, d5, a0, and a1.                *
                           *                                                                              *
                           *       Interrupts:                                                            *
                           *               This does its own enabling/disabling.  Interrupts are left ON. *
                           *                                                                              *
                           *       This routine uses the card's SEMAPHORE to gain access.                 *
                           *                                                                              *
                           *       This routine calls gain_access, release_access, TXDATABUFFroom,        *
                           *       and find_DATA_AREA.                                                    *
                           *                                                                              *
                           ********************************************************************************
                                                
            0000 099A putchars equ     *
0000099A 6100 F70A             bsr     find_DATA_AREA          ; Setup a1 = data buffer base addr
                           *                                                      d4 = end of data buffer addr
                           *                                                      d5 = TXDATABUFF_SIZE
0000099E 6100 F774             bsr     dir
000009A2 61D4                  bsr     TXDATABUFFroom          ; d3.l = available buffer positions
000009A4 2200                  move.l  d0,d1                   ; d0.l = d1.l = TXDATABUFF_FILL
000009A6 2004                  move.l  d4,d0
000009A8 0280 0000             andi.l  #$0000FFFE,d0
             FFFE
000009AE E280                  asr.l   #1,d0                   ; d0.l = unshifted TXDATABUFF_END
000009B0 9081                  sub.l   d1,d0                   ; d0.l = remaining positions to wrap
                                                
000009B2 B680                  cmp.l   d0,d3                   ; If d0>d3 then set d0 := d3
000009B4 6E02                  bgt.s   pc1
000009B6 2003                  move.l  d3,d0
                                                
000009B8 242C 0042 pc1         move.l  data_number(a4),d2      ; Fetch number of chars avail into d2
000009BC B480                  cmp.l   d0,d2                   ; If d0>d2 then set d0 := d2
```

```
000009BE 6E02              bgt.s   pc2
000009C0 2002              move.l  d2,d0

000009C2 2600       pc2    move.l  d0,d3                   ; d3.l saves number of chars actually
                    *                                      ; transferred below
000009C4 6740              beq.s   pcdone                  ; If zero, no work to be done
000009C6 5340              subq.w  #1,d0                   ; Make offset correct for dbf instr.
000009C8 206C 003E         movea.l data_address(a4),a0     ; Get character pointer into a0

000009CC E349              lsl.w   #1,d1
000009CE D28B              add.l   a3,d1
000009D0 48E7 0040         movem.l a1,-(sp)                ; Save a1 so we can use the register
000009D4 2241              movea.l d1,a1                   ; Now a1 is useable pointer

000009D6 1298       pcloop move.b  (a0)+,(a1)              ; Transfer a character & bump source ptr
000009D8 5449              addq.w  #2,a1                   ; Bump destination pointer (odd bytes)
000009DA 51C8 FFFA         dbf     d0,pcloop               ; Then decrement d0 & loop

000009DE 9483              sub.l   d3,d2
000009E0 2942 0042         move.l  d2,data_number(a4)      ; Now store adjusted number and
000009E4 2948 003E         move.l  a0,data_address(a4)     ; address fields

000009E8 2209              move.l  a1,d1                   ; Move 68000 FILL pointer into d1
000009EA 4CDF 0200         movem.l (sp)+,a1                ; Restore a1 before we forget!
000009EE B881              cmp.l   d1,d4                   ; Now check to see if FILL was moved
000009F0 6602              bne.s   pc3                     ; past end of buffer.  If so, set to
000009F2 2209              move.l  a1,d1                   ; the front of the buffer.
000009F4 0881 0000 pc3     bclr    #0,d1                   ; Fix up the 68000 pointer to be the
000009F8 EF59              rol.w   #7,d1                   ; card's type of pointer
000009FA 6100 F62E         bsr     gain_access
000009FE 038A 0018         movep.w d1,DATA_AREA+FILL(a2)   ; Remember d1 = card's FILL pointer.
00000A02 6100 F648         bsr     release_access
00000A06 6000 F704 pcdone  bra     eir
```



```
                    ********************************************************************************
                    *                                                                              *
                    *       routine putctrlblk:     Routine which puts a control block into the    *
                    *               ==========      Transmit buffer area of the card.  The         *
                    *                               appropriate pointers are updated to reflect    *
                    *                               the control block.  This routine also contains *
                    *                               a timeout mechanism which will be adjusted     *
                    *                               to the proper values later.  If a timeout      *
                    *                               occurs, an escape is done with NO DAMAGE to    *
                    *                               the buffer.  The only thing that can cause the *
                    *                               timeout is < 4 positions left in the control   *
                    *                               buffer.  SEMAPHORE timeout is not handled      *
                    *                               by this routine.                               *
                    *                                                                              *
                    *       At entry:                                                              *
                    *               sc_subtabletype.term =  TERM field for control block (8 bits)  *
                    *               sc_subtabletype.mode =  MODE field for control block (8 bits)  *
                    *               a2.l =  TX buffer record base address (shifted, +1+selectcode) *
                    *               a3.l =  card base address ($00xx0001)                          *
                    *               a4.l =  pointer to sc_subtabletype structure                   *
                    *                                                                              *
                    *       Upon exit:                                                             *
                    *               FILL in the card's transmit control buffer is updated.         *
                    *               a1, d4 and d5 are left with the values from find_CTRL_AREA.    *
                    *               This bashes d0, d1, d2, d3, d4, d5, a0 and a1.                 *
                    *               This uses inner/outer_tx_count for computing timeouts.         *
                    *                                                                              *
                    *       Interrupts:                                                            *
                    *               This does its own enabling/disabling.  Interrupts are left ON. *
                    *                                                                              *
                    *       This routine uses the card's SEMAPHORE to gain access.                 *
                    *                                                                              *
                    *       This routine calls TXCTRLBUFFroom, escape, gain_access, find_CTRL_AREA,*
                    *       eir, dir and release_access.                                           *
                    *                                                                              *
                    ********************************************************************************

              0000 0A0A putctrlblk equ  *
00000A0A 6100 F6A2         bsr     find_CTRL_AREA          ; Setup a1 = ctrl buffer base addr
                    *                                                  d4 = end of ctrl buffer addr
                    *                                                  d5 = TXCTRLBUFF_SIZE
00000A0E 296C 002E         move.l  timeout(a4),outer_tx_count(a4)
                  0046
00000A14 297C 0000         move.l  #pcbtimeout,inner_tx_count(a4) ; Load timeout value
                  000B 0052
00000A1C 6100 F6F6 pcbwait bsr     dir
00000A20 6100 FF44         bsr     TXCTRLBUFFroom          ; Get d3 = #bytes available in buffer
00000A24 0C83 0000         cmpi.l  #4,d3                   ; and d0 = CTRLBUFF_FILL (unshifted)
                  0004
00000A2A 6C22              bge.s   roomok                  ; If >=4 bytes, can go ahead!
00000A2C 6100 F6DE         bsr     eir
00000A30 53AC 0052         subq.l  #1,inner_tx_count(a4)
00000A34 66E6              bne.s   pcbwait                 ; Loop, then if it times out give an
00000A36 297C 0000         move.l  #pcbtimeout,inner_tx_count(a4)
                  000B 0052
00000A3E 4AAC 0046         tst.l   outer_tx_count(a4)
00000A42 67D8              beq.s   pcbwait
```

```
00000A44 53AC 0046          subq.l  #1,outer_tx_count(a4)
00000A48 66D2               bne.s   pcbwait
00000A4A 6000 F870          bra     time_err                ; escape(timeout).

00000A4E E348       roomok  lsl.w   #1,d0                   ; Make CTRLBUFF_FILL into a 68000
00000A50 D08B               add.l   a3,d0                   ; pointer
00000A52 2040               movea.l d0,a0                   ; Put in a0 to use it.

00000A54 010A 0018          movep.w DATA_AREA+FILL(a2),d0   ; Get the DATA_FILL pointer to put
00000A58 0188 0000          movep.w d0,POINTER(a0)          ; into the POINTER FIELD
00000A5C 302C 003C          move.w  term_and_mode(a4),d0
00000A60 0188 0004          movep.w d0,TERMFIELD(a0)
00000A64 D1FC 0000          adda.l  #CTRLBLKSIZE,a0         ; Bump pointer by TWO bytes
              0008

00000A6A 2208               move.l  a0,d1                   ; Move 68000 FILL pointer into d1
00000A6C B881               cmp.l   d1,d4                   ; Now check to see if FILL was moved
00000A6E 6602               bne.s   pcb1                    ; past end of buffer.  If so, set to
00000A70 2209               move.l  a1,d1                   ; the front of the buffer.
00000A72 0881 0000 pcb1     bclr    #0,d1                   ; Fix up the 68000 pointer to be the
00000A76 EF59               rol.w   #7,d1                   ; card's type of pointer
00000A78 6100 F5B0          bsr     gain_access
00000A7C 038A 0008          movep.w d1,CTRL_AREA+FILL(a2)
00000A80 6100 F5CA          bsr     release_access
00000A84 6000 F686          bra     eir
```



```
                            ************************************************
                            *                                              *
                            *   routine try_sending_EOF:  tries to send EOF *
                            *                and returns immediately if      *
                            *                unsuccessful.                   *
                            *                                                *
                            *   At entry:                                    *
                            *       a2.l =  TX buffer-record base addr       *
                            *       a3.l =  card base address                *
                            *       a4.l =  pointer to sc_subtabletype       *
                            *                                                *
                            *   Upon exit:                                   *
                            *       d0.l = 0 if unsuccessful;                *
                            *              1 if successful.                  *
                            *                                                *
                            *   This bashes d0,d1,d2,d3,d4,d5,a0 & a1        *
                            *                                                *
                            ************************************************

              0000 0A88 try_sending_EOF equ *
00000A88 7204               moveq   #4,d1
00000A8A 4280               clr.l   d0
00000A8C 102A FFFC          move.b  TXENDBLOCKSPACE-TXBUFF(a2),d0
00000A90 671C               beq.s   sE3                     ; If it's zero jump down & wait for 4
00000A92 2200               move.l  d0,d1                   ; bytes in the control queue
00000A94 6100 F610 sE1      bsr     find_DATA_AREA          ; Setup a1 = data buffer base addr
                            *                                       d4 = end of data buffer addr
                            *                                       d5 = TXDATABUFF_SIZE
00000A98 6100 F67A sE1loop  bsr     dir
00000A9C 6100 FEDA          bsr     TXDATABUFFroom          ; Now hang until enough space becomes
00000AA0 B681               cmp.l   d1,d3                   ; available in the data queue
00000AA2 6C08               bge.s   sE2
00000AA4 6100 F666          bsr     eir
00000AA8 4280       noroom  clr.l   d0
00000AAA 4E75               rts

00000AAC 7208       sE2     moveq   #8,d1                   ; if TXENDBLOCKSPACE#0 then wait for
                            *                               ; 8 bytes, not 4.
00000AAE 6100 F5FE sE3      bsr     find_CTRL_AREA          ; Setup a1 = ctrl buffer base addr
                            *                                       d4 = end of ctrl buffer addr
                            *                                       d5 = TXCTRLBUFF_SIZE
00000AB2 6100 F660 sE3loop  bsr     dir
00000AB6 6100 FEAE          bsr     TXCTRLBUFFroom          ; Now hang until enough space becomes
00000ABA B681               cmp.l   d1,d3                   ; available in the ctrl queue
00000ABC 6C06               bge.s   sE4
00000ABE 6100 F64C          bsr     eir
00000AC2 60E4               bra.s   noroom

00000AC4 4A2A FFFC sE4      tst.b   TXENDBLOCKSPACE-TXBUFF(a2)  ; There's enough room now!! If zero
00000AC8 6726               beq.s   sE6                     ; then just send 1 block below
00000ACA 397C 0501          move.w  #$0501,term_and_mode(a4)
              003C
00000AD0 6100 FF38          bsr     putctrlblk
00000AD4 42AC 0042          clr.l   data_number(a4)         ; Followed by some space
00000AD8 196A FFFC          move.b  TXENDBLOCKSPACE-TXBUFF(a2),data_number+3(a4)
              0045
```

```
2729    00000ADE 297C FFFF         move.l  #$FFFF0000,data_address(a4) ; kluge so it isn't left pointing
                 0000 003E
2730                        *                                          to nowhere & get bus error!
2731    00000AE6 6100 FEB2 sE5     bsr     putchars
2732    00000AEA 4AAC 0042         tst.l   data_number(a4)            ; Hang until all sent
2733    00000AEE 66F6              bne.s   sE5
2734    00000AF0 397C 0500 sE6     move.w  #$0500,term_and_mode(a4)
                 003C
2735    00000AF6 6100 FF12         bsr     putctrlblk
2736    00000AFA 7001              moveq   #1,d0
2737    00000AFC 4E75              rts
2738
2739
2740                               end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

DISCINT

Purpose

DISCINT contains assembly language low-level drivers.

Usage

DISCINT is used for the 98625 interface.

Requirements

DI_DRV and COMASM

```
 PAGE   1 [2.0] 11/4/82 16:21:33  ASSEMBLY OF DISCINT.TEXT
PASS 1 COMPLETE. ERRORS: 0
```

```
PAGE   2 [2.0] 11/4/82 16:21:33 IOLIB EXTDI - 98625 DISC INTERFACE DRIVERS

                        ********************************************************************************
                        *
                        *       copyright (c) 1982 by HEWLETT-PACKARD COMPANY
                        *
                        ********************************************************************************
                        *
                        *
                        *       iolib     extdi
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *
                        *       library - iolib
                        *
                        *       purpose - this set of assembly language code is intended to be used as
                        *                 a PASCAL module for i/o drivers for use by the external i/o
                        *                 procedures library.
                        *
                        *       date    - 05/03/82
                        *       update  - 08/10/82
                        *       release - 09/22/82
                        *
                        *
                        *       source  - IOLIB:DISCINT.TEXT
                        *       object  - IOLIB:DISCINT.CODE
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *       released
                        *       version         2.0
                        *
                        *
                        ********************************************************************************
```

```
41                      ********************************************************************************
42                      *
43                      *
44                      *       the following lines are used to tell the linker/loader what this module
45                      *       looks like in PASCAL terms.
46                      *
47                      *       note that it is possible to create assembly modules that are functions.
48                      *       these routines are called through an indirect pointer using the call
49                      *       facility which does not permit functions.
50                      *
51                      *       this module is called 'extdi' ( upper or lower case - doesn't matter )
52                      *       independent of the file name ( by use of the mname pseudo-op ).
53                      *
54                      *       all the externally used procedures are called 'extdi_@@@@@@@' in
55                      *       this module.  if you are using assembly to access them use the
56                      *       'extdi_@@@@@@@' name.  if you are using PASCAL use the '@@@@@@@'
57                      *       name.
58                      *
59                      ********************************************************************************
60
61                       mname extdi
62
63                       src module extdi;
64                       src import iodeclarations;
65                       src export
66                       src         procedure edi_init  ( temp : anyptr );
67                       src         procedure edi_isr   ( temp : anyptr );
68                       src         procedure edi_rdb   ( temp : anyptr ;  var x : char);
69                       src         procedure edi_wtb   ( temp : anyptr ;  val   : char);
70                       src         procedure edi_rdw   ( temp : anyptr ;  var x : io_word);
71                       src         procedure edi_wtw   ( temp : anyptr ;  val   : io_word);
72                       src         procedure edi_rds   ( temp : anyptr ;  reg   : io_word;
73                       src                                                var x : io_word);
74                       src         procedure edi_wtc   ( temp : anyptr ;  reg   : io_word;
75                       src                                                val   : io_word );
76                       src         procedure edi_tfr   ( temp : anyptr ;  bcb   : anyptr );
77                       src         procedure edi_send  ( temp : anyptr ;  val   : char );
78                       src         procedure edi_end   ( temp : anyptr ;  var x : boolean );
79                       src         procedure edi_ppoll ( temp : anyptr ;  var x : char );
80                       src         procedure edi_clr   ( temp : anyptr ;  line  : io_bit );
81                       src         procedure edi_set   ( temp : anyptr ;  line  : io_bit );
82                       src         procedure edi_test  ( temp : anyptr ;  line  : io_bit ;
83                       src                                                var x : boolean );
84                       src
85                       src end; { of extdi )
```



```
87                      ********************************************************************************
88                      *
89                      *       symbols for export as procedure names
90                      *
91                      ********************************************************************************
92
93                                def     extdi_extdi
94                                def     extdi_edi_init
95                                def     extdi_edi_isr
96                                def     extdi_edi_rdb
97                                def     extdi_edi_wtb
98                                def     extdi_edi_rdw
99                                def     extdi_edi_wtw
100                               def     extdi_edi_rds
101                               def     extdi_edi_wtc
102                               def     extdi_edi_tfr
103
104                               def     extdi_edi_send
105                               def     extdi_edi_ppoll
106                               def     extdi_edi_set
107                               def     extdi_edi_clr
108                               def     extdi_edi_test
109                               def     extdi_edi_end
110
111
112
113                     ********************************************************************************
114                     *
115                     *       symbols for import - common assembly language routines
116                     *
117                     *       the routines are in the module common_assembly
118                     *
119                     ********************************************************************************
120
121                               refa    dropdma         give up DMA resource
122                               refa    getdma          actually get DMA
123                               refa    testdma         check to see if DMA is available
124                               refa    logint          branch to user isr
125                               refa    logeot          branch to user eot
126                               refa    stbsy           set buffer busy
127                               refa    stclr           set buffer not busy
128                               refa    DMA_stbsy       set buffer DMA busy
129                               refa    itxfr           is there a tfr active ?
130                               refa    abort_io        kill any tfr active
131                               refa    wait_tfr        timed wait for tfr active
132                               refa    check_tfr       timed wait for tfr - direction
133
134
135                               lmode   dropdma,getdma,testdma,logint,logeot,stbsy
136                               lmode   stclr,DMA_stbsy,itxfr,abort_io,wait_tfr,check_tfr
137
138
```

```
141                                     include COMDCL
```



```
144                     ********************************************************************************
145                     *
146                     *       modified:  02/22/82  JPC   added parm to user EOT & ISR proc's
147                     *
148                     ********************************************************************************
149                     *
150                     *       HPL CONVENTIONS
151                     *
152                     *
153                     *         Much of this code is taken intact from the 9826 HPL
154                     *         language system EIO ROM ( extended I/O ROM ).
155                     *
156                     *         The Pascal that will be calling this code uses
157                     *         the stack for parameter passage.  The HPL code
158                     *         uses the Ax and Dx registers for all parameters.
159                     *         The Pascal driver entry points on the previous pages
160                     *         take care of getting the parameters into the correct
161                     *         registers.
162                     *
163                     *
164                     *       GENERAL HPL ENTRY/EXIT CONDITIONS:
165                     *
166                     *         A1.L = CARD ADDRESS
167                     *         A2.L = DRIVER TEMP ADDRESS
168                     *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
169                     *
170                     *
171                     *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
172                     *
173                     *         A3.L = BUFFER CONTROL BLOCK ADDRESS
174                     *                In addition to the A1/A2 convention, Pascal will use
175                     *                A3 for a pointer to the buffer control block.
176                     *                The HPL system kept much of the transfer
177                     *                information in the s.c. temps.
178                     *
179                     *         TIMEOUT(A2) = contains timeout information
180                     *                Timeout was a global temp in HPL and a timeout
181                     *                generated an error.
182                     *                In PASCAL each card has a timeout value stored in
183                     *                its temporary area.  A timeout error
184                     *                generates an ESCAPE ( which can be trapped ).
185                     *
186                     *
187                     ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *         DRIVER TEMPS TEMPLATE
                        *
                        *           OFFSET FROM A2
                        *
                        *           HPL DECLARATIONS ( MODIFIED )
                        *
                        *
                        ********************************************************************************
0000 0000 ISR_ENTRY     EQU 0       ..19        PASCAL ISR LINK & UNLINK area
0000 0014 USER_ISR      EQU 20                  user ISR:  do NOT change the proc/stat link/parm ordering!!!
0000 0014 H_ISR_PR      EQU 20      ..23            procedure ptr
0000 0018 H_ISR_SL      EQU 24      ..27            static link
0000 001C H_ISR_PM      EQU 28      ..31            parameter
0000 0020 C_ADR         EQU 32      ..35        card address
0000 0024 BUFI_OFF      EQU 36      ..39        buffer pointer offset
0000 0028 BUFO_OFF      EQU 40      ..43        buffer pointer offset
0000 002C EIRB_OFF      EQU 44                  eir byte
0000 002D IO_SC         EQU 45                  select code ( i.e. 7, 22, etc. )
0000 002E TIMEOUT       EQU 46      ..49        timeout value
                        *                           =0 : no timeout
                        *                           #0 : value of timeout
0000 0032 MA_W          EQU 50      ..51        word access to my address
0000 0033 MA            EQU 51                  byte access to my address
0000 0034 AVAIL_OFF     EQU 52      ..??        standard space taken from temps
                        *               52      ..83        normal cards ( 32  bytes )
                        *               52      ..179       98628  card  ( 128 bytes )
```



```
                        ********************************************************************************
                        *
                        *       TRANSFER OFFSETS IN BUFFER CONTROL BLOCK
                        *
                        *           OFFSET FROM A3
                        *
                        *           PASCAL DECLARATION
                        *
                        ********************************************************************************
0000 0000 TTMP_OFF      EQU 0   ..3         pointer to driver temp offset
0000 0005 T_SC_OFF      EQU 5               transfer select code
0000 0007 TACT_OFF      EQU 7               actual transfer mode
0000 0009 TUSR_OFF      EQU 9               transfer mode
                        *                       00   -                   not used
                        *                       01   serial DMA
                        *                       02   serial FHS
                        *                       03   serial FASTEST ( DMA or FHS )
                        *                       04   -                   not used
                        *                       ---------------
                        *                       05   overlp INTR
                        *                       06   overlp DMA
                        *                       07   overlp FHS      ( BURST )
                        *                       08   overlp FASTEST ( DMA or BURST )
                        *                       09   overlp OVERLAP ( DMA or INTR )
0000 000A T_BW_OFF      EQU 10              transfer byte/word indicator
                        *                       0 = byte   / 1 = word
0000 000B TEND_OFF      EQU 11              transfer EOI/END indicator
                        *                       0 = no eoi / 1 = eoi sent or searched for
0000 000D TDIR_OFF      EQU 13              transfer direction
                        *                       0 = input  /  1 = output
0000 000E TCHR_OFF      EQU 14  ..15        transfer terminate character
                        *                       -1 = no termination character
0000 0010 TCNT_OFF      EQU 16  ..19        transfer count
0000 0014 TBUF_OFF      EQU 20  ..23        transfer buffer pointer
0000 0018 TBSZ_OFF      EQU 24  ..27        transfer buffer maximum size
0000 001C TEMP_OFF      EQU 28  ..31        transfer empty pointer pointer
0000 0020 TFIL_OFF      EQU 32  ..35        transfer fill pointer
0000 0024 T_PR_OFF      EQU 36  ..39        transfer pointer to eot procedure
                        *                   NIL  no procedure
0000 0028 T_SL_OFF      EQU 40  ..43        transfer eot proc static link
0000 002C T_PM_OFF      EQU 44  ..47        transfer eot proc parameter
0000 0030 T_DMAPRI      EQU 48              dma priority request
                        *
                        *       TRANSFER EQUATES
                        *
0000 0001 TT_INT        EQU 1               interrupt
0000 0002 TT_DMA        EQU 2               DMA
0000 0003 TT_BURST      EQU 3               burst
0000 0004 TT_FHS        EQU 4               fast handshake
```

```
                        ********************************************************************************
                        *
                        *       EXTERNAL REFERANCES for escape
                        *
                        ********************************************************************************
                                  REFA iodeclarations           reference the io lib var. area
                                  REFA sysglobals


                        ********************************************************************************
                        *
                        *       Escape code values
                        *
                        ********************************************************************************
0000 0001 NO_CARD   EQU     1                       no interface
0000 0002 NOT_HPIB  EQU     2                       not an hpib interface
0000 0003 NO_ACTL   EQU     3                       no active controller
0000 0004 NO_DVC    EQU     4                       sc ( not device ) specified
0000 0005 NO_SPACE  EQU     5                       not enough space in the buffer
0000 0006 NO_DATA   EQU     6                       not enough data left in the buffer
0000 0007 TFR_ERR   EQU     7                       tfr error
0000 0008 SC_BUSY   EQU     8                       sc is currently busy
0000 0009 BUF_BUSY  EQU     9                       the buffer is busy
0000 000A TCNTERR   EQU     10                      bad count
0000 000B BADTMO    EQU     11                      bad timeout value
0000 000C NO_DRV    EQU     12                      no driver
0000 000D NO_DMA    EQU     13                      no dma installed
0000 000E NO_WORD   EQU     14                      no word transfers allowed
0000 000F NOT_TALK  EQU     15                      not addressed as talker
0000 0010 NOT_LSTN  EQU     16                      not addressed as listener
0000 0011 TMO_ERR   EQU     17                      timeout
0000 0012 NO_SCTL   EQU     18                      not system controller
0000 0013 BAD_RDS   EQU     19                      bad read status / write control
0000 0014 BAD_SCT   EQU     20                      bad set/clear/test
0000 0015 CRD_DWN   EQU     21                      interface is dead
0000 0016 EOD_SEEN  EQU     22                      end of data has happened
0000 0017 IO_MISC   EQU     23                      misc. error


FFFF FFE6 IOE_ERROR EQU    -26                      io sub system error escape code


FFFF FFBE IOE_RSLT  EQU    IODECLARATIONS-66
FFFF FFBA IOE_SC    EQU    IODECLARATIONS-70


FFFF FFFE ESC_CODE  EQU    SYSGLOBALS-2
FFFF FFF6 RCVR_BLK  EQU    SYSGLOBALS-10
```



```
                        *
                        * disc interface card temp definitions
                        *

0000 0034 ABI               equ     avail_off+0     PHI/ABI flag
0000 0035 EOI_in            equ     avail_off+1     EOI flag for previous byte in
0000 0036 EOI_out           equ     avail_off+2     EOI flag for next byte out

0000 0039 flags             equ     avail_off+5     driver flags and status byte 0 mask:
                        *                           bit 0:  pass control flag
                        *                           bit 1:  not used
                        *                           bit 2:  error indicator
                        *                           bit 3:  IFC indicator
                        *                           bit 4:  dcl indicator
                        *                           bit 5:  get indicator
                        *                           bit 6:  current rsv status bit.
                        *                           bit 7:  if set, 9914 is in holdoff mode, therefore
                        *                                   issue release hold off before reading, and
                        *                                   use take control sync to set ATN.

0000 003A ppollmsk          equ     avail_off+6     value to put in di_ppoll when ist = 1
                        *   equ     avail_off+7     value to put in di_ppoll when ist = 0

0000 003C DMA_count         equ     avail_off+8     remaining DMA count
0000 0040 DMA_arm_addr      equ     avail_off+12    arm address of the assigned DMA channel
0000 0044 DMA_arm_word      equ     avail_off+16    arm word of the assigned DMA channel
```

```
                        ********************************************************************************
                        *
                        *         PASCAL driver entry points for disc interface cards
                        *
                        ********************************************************************************

                        *
                        * module initialization
                        *
00000000 4E75           extdi_extdi     rts                             do nothing

                        *
                        * driver initialization
                        *
00000002 205F           extdi_edi_init  movea.l (sp)+,a0                get return address
00000004 245F                           movea.l (sp)+,a2                get temp address
00000006 226A 0020                      movea.l c_adr(a2),a1            get card address
0000000A 4850                           pea     (a0)                    push return address back on stack
0000000C 6000 0110                      bra     di_init

                        *
                        * interrupt service routine
                        *
00000010 205F           extdi_edi_isr   movea.l (sp)+,a0                get return address
00000012 245F                           movea.l (sp)+,a2                get temp address
00000014 226A 0020                      movea.l c_adr(a2),a1            get card address
00000018 4850                           pea     (a0)                    push return address back on stack
0000001A 6000 03D2                      bra     di_isr

                        *
                        * read a byte
                        *
0000001E 205F           extdi_edi_rdb   movea.l (sp)+,a0                get return address
00000020 265F                           movea.l (sp)+,a3                get var address
00000022 245F                           movea.l (sp)+,a2                get temp address
00000024 226A 0020                      movea.l c_adr(a2),a1            get card address
00000028 4850                           pea     (a0)                    push return address back on stack
0000002A 6100 014E                      bsr     di_rdb                  call read byte
0000002E 1680                           move.b  d0,(a3)                 save character
00000030 4E75                           rts

                        *
                        * write a byte
                        *
00000032 205F           extdi_edi_wtb   movea.l (sp)+,a0                get return address
00000034 101F                           move.b  (sp)+,d0                get value
00000036 245F                           movea.l (sp)+,a2                get temp address
00000038 226A 0020                      movea.l c_adr(a2),a1            get card address
0000003C 4850                           pea     (a0)                    push return address back on stack
0000003E 6000 0168                      bra     di_wtb                  call write byte
```



```
                        *
                        * read a word
                        *
00000042 205F           extdi_edi_rdw   movea.l (sp)+,a0                get return address
00000044 265F                           movea.l (sp)+,a3                get var address
00000046 245F                           movea.l (sp)+,a2                get temp address
00000048 226A 0020                      movea.l c_adr(a2),a1            get card address
0000004C 4850                           pea     (a0)                    push return address back on stack
0000004E 6100 012A                      bsr     di_rdb                  read first byte
00000052 16C0                           move.b  d0,(a3)+                save first byte
00000054 6100 0124                      bsr     di_rdb                  read second byte
00000058 1680                           move.b  d0,(a3)                 save second byte
0000005A 4E75                           rts

                        *
                        * write a word
                        *
0000005C 205F           extdi_edi_wtw   movea.l (sp)+,a0                get return address
0000005E 301F                           move    (sp)+,d0                get word value
00000060 245F                           movea.l (sp)+,a2                get temp address
00000062 226A 0020                      movea.l c_adr(a2),a1            get card address
00000066 4850                           pea     (a0)                    push return address back on stack
00000068 1A00                           move.b  d0,d5                   save second byte
0000006A E048                           lsr     #8,d0
0000006C 6100 013A                      bsr     di_wtb                  write the byte
00000070 1005                           move.b  d5,d0                   get the second byte
00000072 6000 0134                      bra     di_wtb                  write the byte

                        *
                        * read status
                        *
00000076 205F           extdi_edi_rds   movea.l (sp)+,a0                get return address
00000078 265F                           movea.l (sp)+,a3                get var address
0000007A 321F                           move    (sp)+,d1                get register number
0000007C 245F                           movea.l (sp)+,a2                get temp address
0000007E 226A 0020                      movea.l c_adr(a2),a1            get card address
00000082 4850                           pea     (a0)                    push return address back on stack
00000084 6100 01C4                      bsr     di_rds                  read status
00000088 3680                           move    d0,(a3)                 save status info
0000008A 4E75                           rts

                        *
                        * write control
                        *
0000008C 205F           extdi_edi_wtc   movea.l (sp)+,a0                get return address
0000008E 301F                           move    (sp)+,d0                get value
00000090 321F                           move    (sp)+,d1                get register number
00000092 245F                           movea.l (sp)+,a2                get temp address
00000094 226A 0020                      movea.l c_adr(a2),a1            get card address
00000098 4850                           pea     (a0)                    push return address back on stack
0000009A 6000 020E                      bra     di_wtc                  write control
```

```
                        *
                        * transfer
                        *
0000009E 205F           extdi_edi_tfr   movea.l (sp)+,a0              get return address
000000A0 265F                           movea.l (sp)+,a3              get buffer control block address
000000A2 245F                           movea.l (sp)+,a2              get temp address
000000A4 226A 0020                      movea.l c_adr(a2),a1          get card address
000000A8 4850                           pea     (a0)                  push return address back on stack
000000AA 6000 0494                      bra     di_tfr                transfer

                        *
                        * send an 'ATN' true command
                        *
000000AE 205F           extdi_edi_send  movea.l (sp)+,a0              get return address
000000B0 101F                           move.b  (sp)+,d0              get value
000000B2 245F                           movea.l (sp)+,a2              get temp address
000000B4 226A 0020                      movea.l c_adr(a2),a1          get card address
000000B8 4850                           pea     (a0)                  push return address back on stack
000000BA 6000 0312                      bra     di_r6out              send command byte

                        *
                        * perform a parallel poll
                        *
000000BE 205F           extdi_edi_ppoll movea.l (sp)+,a0              get return address
000000C0 265F                           movea.l (sp)+,a3              get var address
000000C2 245F                           movea.l (sp)+,a2              get temp address
000000C4 226A 0020                      movea.l c_adr(a2),a1          get card address
000000C8 4850                           pea     (a0)                  push return address back on stack
000000CA 6100 0234                      bsr     di_p_poll             do a parallel poll
000000CE 1680                           move.b  d0,(a3)               save value
000000D0 4E75                           rts

                        *
                        * set an hpib line
                        *
000000D2 205F           extdi_edi_set   movea.l (sp)+,a0              get return address
000000D4 321F                           move    (sp)+,d1              get line
000000D6 245F                           movea.l (sp)+,a2              get temp address
000000D8 226A 0020                      movea.l c_adr(a2),a1          get card address
000000DC 4850                           pea     (a0)                  push return address back on stack
000000DE 6000 0256                      bra     di_set                call set line

                        *
                        * clear an hpib line
                        *
000000E2 205F           extdi_edi_clr   movea.l (sp)+,a0              get return address
000000E4 321F                           move    (sp)+,d1              get line
000000E6 245F                           movea.l (sp)+,a2              get temp address
000000E8 226A 0020                      movea.l c_adr(a2),a1          get card address
000000EC 4850                           pea     (a0)                  push return address back on stack
000000EE 6000 02A6                      bra     di_clr                clear the line
```



```
                        *
                        * test an hpib line
                        *
000000F2 205F           extdi_edi_test  movea.l (sp)+,a0              get return address
000000F4 265F                           movea.l (sp)+,a3              get var address
000000F6 321F                           move    (sp)+,d1              get line
000000F8 245F                           movea.l (sp)+,a2              get temp address
000000FA 226A 0020                      movea.l c_adr(a2),a1          get card address
000000FE 4850                           pea     (a0)                  push return address back on stack
00000100 6100 02C8                      bsr     di_test               read status
00000104 1680                           move.b  d0,(a3)               save character
00000106 4E75                           rts

                        *
                        * test for EOI/end condition
                        *
00000108 205F           extdi_edi_end   movea.l (sp)+,a0              get return address
0000010A 265F                           movea.l (sp)+,a3              get var address
0000010C 245F                           movea.l (sp)+,a2              get temp address
0000010E 226A 0020                      movea.l c_adr(a2),a1          get card address
00000112 4850                           pea     (a0)                  push return address back on stack
00000114 102A 0035                      move.b  EOI_in(a2),d0         get eor flag byte
00000118 EE08                           lsr.b   #7,d0                 PASCAL needs it [0..1]
0000011A 1680                           move.b  d0,(a3)               save condition
0000011C 4E75                           rts
```

```
                        ********************************************************************************
                        *
                        *       disc interface card address equates  ( offsets from a1 )
                        *
                        ********************************************************************************

              0000 0001 cardid          equ $01   read          card identification
              0000 0001 cardreset       equ $01   write         card software reset
              0000 0003 cardstatus      equ $03   read          card status
              0000 0003 cardcontrol     equ $03   write         card control
              0000 0007 cardlatch       equ $07   read/write    latch for testing card buffers

              0000 0011 intreg          equ $11   read/write    PHI/ABI interrupt register
              0000 0013 intmask         equ $13   read/write    PHI/ABI interrupt mask
              0000 0015 fifo            equ $15   read/write    PHI/ABI inbound/outbound fifo
              0000 0017 status          equ $17   read/write    PHI/ABI status register
              0000 0019 control         equ $19   read/write    PHI/ABI control register
              0000 001B address         equ $1B   read/write    PHI/ABI hp-ib address register
              0000 001D ppolmask        equ $1D   read/write    PHI/ABI parallel poll mask register
              0000 001F ppolsense       equ $1F   read/write    PHI/ABI parallel poll sense register


                        ********************************************************************************
                        *
                        *       hp-ib command equates
                        *
                        ********************************************************************************

              0000 0001 gtl             equ     1               go to local
              0000 0004 sdc             equ     4               selective device clear
              0000 0005 ppc             equ     5               ppoll configure
              0000 0008 get             equ     8               group execute trigger
              0000 0009 tct             equ     9               take control
              0000 0011 llo             equ     17              local lockout
              0000 0014 dcl             equ     20              device clear
              0000 0015 ppu             equ     21              ppoll unconfigure
              0000 0018 spe             equ     24              spoll enable
              0000 0019 spd             equ     25              spoll disable
              0000 003F unl             equ     63              unlisten
              0000 005F unt             equ     95              untalk
              0000 0060 ppe             equ     96              ppoll enable
              0000 0070 ppd             equ     112             ppoll disable
```



```
                        ********************************************************************************
                        *
                        *       di_init
                        *
                        *         initialize a disc interface card
                        *
                        ********************************************************************************

    0000011E 701E       di_init         moveq   #30,d0                  my address if active controller
    00000120 6106                       bsr.s   di_init_s               start software reset
    00000122 6600 0246                  bne     di_IFC                  if system controller, branch
    00000126 4E75                       rts


                        ********************************************************************************
                        *
                        *       di_int_s
                        *
                        *         subroutine used for both initialization and wtc:
                        *
                        ********************************************************************************

                        *
                        * software reset the card
                        *
    00000128 7280       di_init_s       moveq   #$80,d1                 prepare to...
    0000012A 1341 0001                  move.b  d1,cardreset(a1)        software reset the card (PHI/ABI pon)
    0000012E 1341 0019                  move.b  d1,control(a1)          set 8-bit mode
    00000132 1341 0019                  move.b  d1,control(a1)          once more, to guarantee high-order bit values
    00000136 3540 0032                  move    d0,ma_w(a2)             save my address
    0000013A D001                       add.b   d1,d0                   set the "online" bit
    0000013C 1341 0017                  move.b  d1,status(a1)           prepare to enable CRC if ABI
    00000140 1340 001B                  move.b  d0,address(a1)          bring PHI/ABI online with specified address
    00000144 4A29 001B                  tst.b   address(a1)             is this an ABI?
    00000148 0829 0007                  btst    #7,status(a1)           a writeable CRC bit will tell
             0017
    0000014E 56EA 0034                  sne     ABI(a2)                 remember it
                        *
                        * set up the interrupt register & driver temps
                        *
    00000152 4EB9 0000                  jsr     abort_io                cleanup any attached buffer
             0000
    00000158 51EA 0035                  sf      EOI_in(a2)              clear the EOI in flag
    0000015C 51EA 0036                  sf      EOI_out(a2)             clear the EOI out flag
    00000160 41E9 0017                  lea     status(a1),a0           point to the status register
    00000164 10BC 0080                  move.b  #$80,(a0)               set high-order bits
    00000168 137C 0000                  move.b  #$00,intmask(a1)        initial interrupt mask ????????????????
             0013
    0000016E 137C 0080                  move.b  #$80,cardcontrol(a1)    enable card interrupts
             0003

    00000174 0810 0003                  btst    #3,(a0)                 is this THE system controller?
    00000178 4E75                       rts                             (leave cc for caller)
```

```
                        ********************************************************************************
                        *
                        *       di_rdb
                        *
                        *         read a byte of data from hp-ib
                        *
                        *         exit:   d0.l = byte read
                        *
                        ********************************************************************************

0000017A 41E9 0017      di_rdb          lea     status(a1),a0           point to the status register
0000017E 0810 0001                      btst    #1,(a0)                 make sure addressed to listen
00000182 675E                           beq.s   di_lsterr               else give error
00000184 0810 0004                      btst    #4,(a0)                 active controller?
00000188 670A                           beq.s   di_rdb1                 branch if not
0000018A 10BC 0080                      move.b  #$80,(a0)               inhibit LF detection
0000018E 137C 0001      0015            move.b  #1,fifo(a1)             enable a 1 byte counted transfer
00000194 616E           di_rdb1         bsr.s   di_wait_fb              now wait for fifo byte
00000196 7000                           moveq   #0,d0                   clear upper part of d0
00000198 1029 0015                      move.b  fifo(a1),d0             and put data in lower byte
0000019C 0829 0006      0017            btst    #6,status(a1)           tagged with EOI?
000001A2 56EA 0035                      sne     EOI_in(a2)              remember it
000001A6 4E75                           rts                             done!


                        ********************************************************************************
                        *
                        *       di_wtb
                        *
                        *         write a byte of data to hp-ib
                        *
                        *         entry:  d0.b = byte to write
                        *
                        *         hpl routine
                        *
                        ********************************************************************************

000001A8 0829 0002      di_wtb          btst    #2,status(a1)           make sure addressed to talk
      0017
000001AE 6736                           beq.s   di_tlkerr               else error
000001B0 614E                           bsr.s   di_wait_fi              wait for fifo idle
000001B2 142A 0036                      move.b  EOI_out(a2),d2          EOI out flag
000001B6 EF0A                           lsl.b   #7,d2                   prepare to...
000001B8 1342 0017                      move.b  d2,status(a1)           set the high-order bits
000001BC 1340 0015                      move.b  d0,fifo(a1)             move the data out
000001C0 51EA 0036                      sf      EOI_out(a2)             clear the EOI out flag
000001C4 4E75                           rts                             done!
```



```
                        ********************************************************************************
                        *
                        *       error escapes
                        *
                        ********************************************************************************

000001C6 7008           di_scbsy        moveq   #sc_busy,d0             buffer is busy
000001C8 6022                           bra.s   esc_err
000001CA 7014           di_sc_err       moveq   #bad_sct,d0             bad set/clear/test
000001CC 601E                           bra.s   esc_err
000001CE 7003           di_notactl      moveq   #no_actl,d0             not active controller
000001D0 601A                           bra.s   esc_err
000001D2 7012           di_notsctl      moveq   #no_sctl,d0             not system controller
000001D4 6016                           bra.s   esc_err
000001D6 7007           hterr_b         moveq   #tfr_err,d0             bad transfer specification
000001D8 6012                           bra.s   esc_err
000001DA 700D           hterr_d         moveq   #no_DMA,d0              DMA not installed
000001DC 600E                           bra.s   esc_err
000001DE 700E           di_noword       moveq   #no_word,d0             word transfers not allowed
000001E0 600A                           bra.s   esc_err
000001E2 7010           di_lsterr       moveq   #not_lstn,d0            not addressed as listener
000001E4 6006                           bra.s   esc_err
000001E6 700F           di_tlkerr       moveq   #not_talk,d0            not addressed as talker
000001E8 6002                           bra.s   esc_err
000001EA 7011           di_tmo          moveq   #tmo_err,d0             timeout
                        *               bra.s   esc_err
000001EC 2B40 FFBE      esc_err         move.l  d0,ioe_rslt(a5)          save error in io space
000001F0 102A 002D                      move.b  io_sc(a2),d0            \ get sc for error
000001F4 2B40 FFBA                      move.l  d0,ioe_sc(a5)           /
000001F8 3B7C FFE6      FFFE            move    #ioe_error,esc_code(a5) save system esc code
000001FE 4E4A                           trap    #10                     escape
```

```
                        ********************************************************************************
                        *
                        * wait for fifo idle or fifo byte routines
                        *
                        ********************************************************************************

00000200 7201           di_wait_fi      moveq   #1,d1                   fifo idle is bit 1
00000202 6002                           bra.s   di_wfc

00000204 7202           di_wait_fb      moveq   #2,d1                   fifo byte is bit 2

                        *
                        * generalized wait for condition routine
                        *
00000206 4229 0003      di_wfc          clr.b   cardcontrol(a1)         disable card interrupts
0000020A 03E9 0013                      bset    d1,intmask(a1)          unmask the appropriate interrupt bit
0000020E 41E9 0011                      lea     intreg(a1),a0           point to the interrupt register
                        *
                        * quick low-overhead loop
                        *
00000212 747F                           moveq   #127,d2                 quick loop counter
00000214 0310           di_wfc_quick    btst    d1,(a0)                 condition met?
00000216 6626                           bne.s   di_wfc_done             branch if so
00000218 51CA FFFA                      dbra    d2,di_wfc_quick         loop until quick count expires
                        *
                        * timed wait loop
                        *
0000021C 242A 002E                      move.l  timeout(a2),d2          current timeout
00000220 6718                           beq.s   di_wfc_infinite         branch if infinite timeout
00000222 C4FC 01EB                      mulu    #491,d2                 loop iterations per millisecond

00000226 0310           di_wfc_timed    btst    d1,(a0)                 condition met?
00000228 6614                           bne.s   di_wfc_done             branch if so
0000022A 5382                           subq.l  #1,d2                   decrement the loop counter
0000022C 62F8                           bhi     di_wfc_timed            loop until count expired

0000022E 03A9 0013                      bclr    d1,intmask(a1)          re-mask the appropriate interrupt bit
00000232 137C 0080                      move.b  #$80,cardcontrol(a1)    re-enable card interrupts
              0003
00000238 60B0                           bra     di_tmo                  escape with timeout error
                        *
                        * infinite wait loop
                        *
0000023A 0310           di_wfc_infinite btst    d1,(a0)                 condition met?
0000023C 67FC                           beq     di_wfc_infinite         loop until so
                        *
                        * wait for condition done
                        *
0000023E 03A9 0013      di_wfc_done     bclr    d1,intmask(a1)          re-mask the appropriate interrupt bit
00000242 137C 0080                      move.b  #$80,cardcontrol(a1)    re-enable card interrupts
              0003
00000248 4E75                           rts                             done!
```



```
                        ********************************************************************************
                        *
                        *       di_rds
                        *
                        *         read status
                        *
                        *         PASCAL routine
                        *
                        ********************************************************************************

0000024A B27C 0008      di_rds          cmp     #8,d1                   register within range?
0000024E 621C                           bhi.s   rds_err                 branch if not
00000250 D241                           add     d1,d1                   two bytes per table entry
00000252 323B 1006                      move    rds_table(d1),d1        load routine offset
00000256 4EFB 1002                      jmp     rds_table(d1)           jump to the appropriate routine

                        *
                        * rds jump table
                        *
0000025A 0018           rds_table       dc      rds_0-rds_table         status 0
0000025C 0012                           dc      rds_1-rds_table         status 1
0000025E 0012                           dc      rds_2-rds_table         status 2
00000260 001C                           dc      rds_3-rds_table         status 3
00000262 0012                           dc      rds_4-rds_table         status 4
00000264 0012                           dc      rds_5-rds_table         status 5
00000266 0030                           dc      rds_6-rds_table         status 6
00000268 0012                           dc      rds_7-rds_table         status 7
0000026A 0012                           dc      rds_8-rds_table         status 8

          0000 026C     rds_1           equ     *
          0000 026C     rds_2           equ     *
          0000 026C     rds_4           equ     *
          0000 026C     rds_5           equ     *
          0000 026C     rds_7           equ     *
          0000 026C     rds_8           equ     *

0000026C 7013           rds_err         moveq   #bad_rds,d0             bad read status
0000026E 6000 FF7C                      bra     esc_err
```

```
                        *
                        * status 0 - card ID
                        *
00000272 7008           rds_0           moveq   #8,d0                   Simon's ID
00000274 4E75                           rts


                        *
                        * status 3  -  controller status & address
                        *
00000276 7000           rds_3           moveq   #0,d0                   ZZZZZZZZ ZZZZZZZZ
00000278 1029 0017                      move.b  status(a1),d0           ZZZZZZZZ 76543210
0000027C E918                           rol.b   #4,d0                   ZZZZZZZZ 32107654
0000027E E348                           lsl     #1,d0                   ZZZZZZZ3 21076547
00000280 ED08                           lsl.b   #6,d0                   ZZZZZZZ3 4ZZZZZZZ
00000282 E248                           lsr     #1,d0                   ZZZZZZZZ 34ZZZZZZ
00000284 D02A 0033                      add.b   ma(a2),d0               ZZZZZZZZ 34ZAAAAA
00000288 4E75                           rts


                        *
                        * status 6  -  interface status
                        *
0000028A 61EA           rds_6           bsr     rds_3           ZZZZZZZZ ZZZZZZZZ ZZZZZZZZ LLLLLLLL
0000028C E098                           ror.l   #8,d0           LLLLLLLL ZZZZZZZZ ZZZZZZZZ ZZZZZZZZ
0000028E 1029 0017                      move.b  status(a1),d0   LLLLLLLL ZZZZZZZZ ZZZZZZZZ 76543210
00000292 E358                           rol     #1,d0           LLLLLLLL ZZZZZZZZ ZZZZZZZ7 65432107
00000294 E298                           ror.l   #1,d0           ZLLLLLLL LZZZZZZZ ZZZZZZZZ 76543210
00000296 E418                           ror.b   #2,d0           ZLLLLLLL LZZZZZZZ ZZZZZZZZ 10765432
00000298 E298                           ror.l   #1,d0           2ZLLLLLL LLZZZZZZ ZZZZZZZZ Z1076543
0000029A E518                           rol.b   #2,d0           2ZLLLLLL LLZZZZZZ ZZZZZZZZ 076543Z1
0000029C E298                           ror.l   #1,d0           12ZLLLLL LLLZZZZZ ZZZZZZZZ Z076543Z
0000029E E958                           rol     #4,d0           12ZLLLLL LLLZZZZZ ZZZZZ076 543ZZZZZ
000002A0 E898                           ror.l   #4,d0           ZZZZ12ZL LLLLLLLZ ZZZZZZZZ Z076543Z
000002A2 E618                           ror.b   #3,d0           ZZZZ12ZL LLLLLLLZ ZZZZZZZZ 43ZZ0765
000002A4 E298                           ror.l   #1,d0           5ZZZZ12Z LLLLLLLL ZZZZZZZZ Z43ZZ076
000002A6 4840                           swap    d0              ZZZZZZZZ Z43ZZ076 5ZZZZ12Z LLLLLLLL
000002A8 4E75                           rts                             whew!!!
```



```
                        ********************************************************************************
                        *
                        *       di_wtc
                        *
                        *         write control
                        *
                        *         entry:  d0.w = parameter
                        *
                        ********************************************************************************

000002AA 0C41 0001 di_wtc          cmpi    #1,d1                   register within range?
000002AE 62BC                      bhi.s   rds_err                 branch if not
000002B0 D241                      add     d1,d1                   two bytes per table entry
000002B2 4EFB 1002                 jmp     hwtctbl(d1)             jump to the appropriate routine


000002B6 6002           hwtctbl    bra.s   di_wtc_rst              control 0 - do a reset
000002B8 6016                      bra.s   di_rqs                  control 1 - set SRQ response
                        *          bra.s   di_wtc_ppc              control 2 - ppoll configure
                        *          bra.s   di_wtc_sma              control 3 - set my addr
                        *          bra.s   rds_err                 control 4 - not used
                        *          bra.s   rds_err                 control 5 - enable interrupts


                        *
                        * software reset
                        *
000002BA 102A 0033 di_wtc_rst      move.b  ma(a2),d0               else use previous address

000002BE 1F29 0017 hpl_wtc1        move.b  status(a1),-(sp)        save controller active state.
000002C2 6100 FE64                 bsr     di_init_s               do software reset
000002C6 081F 0004                 btst    #4,(sp)+                were we active controller?
000002CA 6702                      beq.s   hpl_wtc2                if not, skip
000002CC 4E71                      nop     ????????????????????????????????????????????????
000002CE 4E75           hpl_wtc2   rts                             else done


                        *
                        * request service
                        *
000002D0 08AA 0006 di_rqs          bclr    #6,flags(a2)            assume rsv = 0 in this new byte
                 0039
000002D6 1200                      move.b  d0,d1                   if rsv bit in the new byte is indeed
000002D8 0881 0006                 bclr    #6,d1                   zero, then just output the new byte.
000002DC 6706                      beq.s   di_rqs2
                        *          move.b  d1,di_spoll(a1)         else first write the byte with rsv
000002DE 08EA 0006                 bset    #6,flags(a2)            clear. remember that rsv is set.
                 0039
000002E4 1280           di_rqs2    move.b  d0,(a1) <<<di_spoll(a1)         write the byte with rsv correct.
000002E6 322A 003A                 move    ppollmsk(a2),d1         go update the parallel poll response
000002EA 4E4B           hpl_wtc4   trap    #11                     get into supervisor mode   scs
                        * scs      move    sr,-(sp)                disable isr's while we figure
000002EC 007C 2700                 ori     #$2700,sr               out which mask to set based on
000002F0 082A 0006                 btst    #6,flags(a2)            current rsv bit.
                 0039
000002F6 6702                      beq.s   hpl_wtc5                if rsv = 0, use right byte
000002F8 E059                      ror     #8,d1                   else use left byte
```

```
 907    000002FA 1281       hpl_wtc5        move.b  d1,(a1) <<<di_ppoll(a1)
 908    000002FC 46DF                       move    (sp)+,sr                restore user mode       scs
 909    000002FE 4E75                       rts                                                     scs
 910                        * scs            rte                            re-enable isr's and return
 911
```



```
 913                        ********************************************************************************
 914                        *
 915                        *       di_p_poll
 916                        *
 917                        *         conduct parallel poll
 918                        *
 919                        *         if not active controller give error
 920                        *         else value returned in d0.b
 921                        *
 922                        *         hpl routine
 923                        *
 924                        ********************************************************************************
 925
 926    00000300 0829 0004 di_p_poll       btst    #4,status(a1)           active controller?
             0017
 927    00000306 6700 FEC6                 beq     di_notactl              branch if not
 928
 929    0000030A 6100 FEF4                 bsr     di_wait_fi              wait for fifo idle
 930
 931    0000030E 4229 0003                 clr.b   cardcontrol(a1)         disable card interrupts
 932    00000312 08E9 0005                 bset    #5,intmask(a1)          unmask the poll response bit for ABI's sake
             0013
 933    00000318 4229 0017                 clr.b   status(a1)              high-order bits
 934    0000031C 50E9 001D                 st      ppolmask(a1)            look at all response bits
 935    00000320 4229 001F                 clr.b   ppolsense(a1)          normal sense for all
 936    00000324 1029 0015                 move.b  fifo(a1),d0             get the response
 937    00000328 08A9 0005                 bclr    #5,intmask(a1)          re-mask the poll response bit
             0013
 938    0000032E 137C 0080                 move.b  #$80,cardcontrol(a1)    re-enable card interrupts
             0003
 939
 940    00000334 4E75                      rts
 941
```

```
                        ********************************************************************************
                        *
                        *       di_set
                        *
                        *         set an hpib line true
                        *
                        *         PASCAL routine
                        *
                        ********************************************************************************

00000336 B27C 0007 di_set          cmp     #7,d1                   register within range?
0000033A 6200 FE8E                 bhi     di_sc_err               branch if not
0000033E D241                      add     d1,d1                   two bytes per table entry
00000340 323B 1006                 move    set_table(d1),d1        routine offset
00000344 4EFB 1002                 jmp     set_table(d1)           jump to the appropriate routine

00000348 0010      set_table       dc      di_REN-set_table        REN  - set REN
0000034A 0022                      dc      di_IFC-set_table        IFC  - pulse IFC (set REN/clr ATN)
0000034C FE82                      dc      di_sc_err-set_table     SRQ  - error
0000034E 0048                      dc      di_EOI-set_table        EOI  - pulse EOI on next byte out
00000350 FE82                      dc      di_sc_err-set_table     NRFD - error
00000352 FE82                      dc      di_sc_err-set_table     NDAC - error
00000354 FE82                      dc      di_sc_err-set_table     DAV  - error
00000356 FE82                      dc      di_sc_err-set_table     ATN  - error


                        *
                        * set REN
                        *
00000358 0829 0003 di_REN          btst    #3,status(a1)           system controller?
         0017
0000035E 6700 FE72                 beq     di_notsctl              branch if not
00000362 08E9 0005                 bset    #5,control(a1)          set REN
         0019
00000368 4E75                      rts


                        *
                        * assert IFC for 100us
                        *
0000036A 0829 0003 di_IFC          btst    #3,status(a1)           system controller?
         0017
00000370 6700 FE60                 beq     di_notsctl              branch if not

00000374 41E9 0019                 lea     control(a1),a0          point to the control register
00000378 0890 0005                 bclr    #5,(a0)                 clear REN
0000037C 08D0 0004                 bset    #4,(a0)                 assert IFC
00000380 7050                      moveq   #80,d0                  must hold IFC at least 100us...
00000382 51C8 FFFE                 dbra    d0,*                    to satisfy the bus standard
00000386 0890 0004                 bclr    #4,(a0)                 release IFC
0000038A 08D0 0005                 bset    #5,(a0)                 set REN

0000038E 4E75                      rts


                        *
                        * assert EOI on the next wtb
```



```
                        *
00000390 50EA 0036 di_EOI          st      EOI_out(a2)             set the flag
00000394 4E75                      rts
```

```
1002                        ********************************************************************************
1003                        *
1004                        *       di_clr
1005                        *
1006                        *         set an hpib line false
1007                        *
1008                        *         PASCAL routine
1009                        *
1010                        ********************************************************************************
1011
1012    00000396 B27C 0007 di_clr          cmp     #7,d1                   register within range?
1013    0000039A 6200 FE2E                 bhi     di_sc_err               branch if not
1014    0000039E D241                      add     d1,d1                   two bytes per table entry
1015    000003A0 323B 1006                 move    clear_table(d1),d1      routine offset
1016    000003A4 4EFB 1002                 jmp     clear_table(d1)         jump to the appropriate routine
1017
1018    000003A8 0010      clear_table     dc      di_local-clear_table    REN  - clear REN
1019    000003AA 0020                      dc      di_dmyrts-clear_table   IFC  - nothing
1020    000003AC FE22                      dc      di_sc_err-clear_table   SRQ  - error
1021    000003AE 0020                      dc      di_dmyrts-clear_table   EOI  - nothing
1022    000003B0 FE22                      dc      di_sc_err-clear_table   NRFD - error
1023    000003B2 FE22                      dc      di_sc_err-clear_table   NDAC - error
1024    000003B4 FE22                      dc      di_sc_err-clear_table   DAV  - error
1025    000003B6 FE22                      dc      di_sc_err-clear_table   ATN  - error
1026
1027
1028                        *
1029                        * clear REN
1030                        *
1031    000003B8 0829 0003 di_local        btst    #3,status(a1)           system controller?
                0017
1032    000003BE 6700 FE12                 beq     di_notsctl              branch if not
1033    000003C2 08A9 0005                 bclr    #5,control(a1)          clear REN
                0019
1034    000003C8 4E75      di_dmyrts       rts
1035
```



```
1037                        ********************************************************************************
1038                        *
1039                        *       di_test
1040                        *
1041                        *         get an hpib line's state
1042                        *
1043                        *         entry :       d1 = line parameter
1044                        *                            0 = REN
1045                        *                            1 = IFC
1046                        *                            2 = SRQ
1047                        *                            3 = EOI
1048                        *                            4 = NRFD
1049                        *                            5 = NDAC
1050                        *                            6 = DAV
1051                        *                            7 = ATN
1052                        *
1053                        *         PASCAL routine
1054                        *
1055                        ********************************************************************************
1056
1057    000003CA 6000 FDFE di_test         bra             di_sc_err
1058
```

```
1060                        ********************************************************************************
1061                        *
1062                        *       di_r6out
1063                        *
1064                        *         emulation of r6 out for hp-ib
1065                        *
1066                        *         entry:  d0 = byte to output
1067                        *
1068                        *         exit:   if not active controller, sts cleared and error bit set
1069                        *                 else operation is done and any addressing decoded.
1070                        *
1071                        *         hpl routine ( modified )
1072                        *
1073                        ********************************************************************************
1074
1075    000003CE 4EB9 0000 di_r6out        jsr     wait_tfr                if a tfr is active wait till it isn't
                0000
1076
1077    000003D4 0829 0004                 btst    #4,status(a1)           active controller?
                0017
1078    000003DA 6700 FDF2                 beq     di_notactl              branch if not
1079
1080    000003DE 6100 FE20                 bsr     di_wait_fi              wait for fifo idle
1081    000003E2 137C 0040                 move.b  #$40,status(a1)         following byte is a command
                0017
1082    000003E8 1340 0015                 move.b  d0,fifo(a1)             send it
1083
1084    000003EC 4E75                      rts
1085
```



```
1088                        ********************************************************************************
1089                        *
1090                        *       di_isr
1091                        *
1092                        *         interrupt service routine for hp-ib cards
1093                        *
1094                        *         entry : a1,a2 are set up
1095                        *
1096                        *         the isr will track down the buffer control block
1097                        *
1098                        *         hpl routine ( modified )
1099                        *
1100                        ********************************************************************************
1101
1102    000003EE 08AA 0001 di_isr          bclr    #1,flags(a2)            clear user isr pending flag
                0039
1103    000003F4 6100 0100                 bsr     di_isr1                 process the interrupting condition(s)
1104    000003F8 4EB9 0000                 jsr     itxfr                   go see if transfer is active
                0000
1105    000003FE 6716                      beq.s   isr_end                 if not, then we are done.
1106    00000400 B23C 0003                 cmp.b   #tt_burst,d1            if frw is active, go process it
1107    00000404 6700 00E2                 beq     di_frw
1108    00000408 B23C 0001                 cmp.b   #tt_int,d1              if int then go process it
1109    0000040C 6700 00C4                 beq     di_buf
1110    00000410 B23C 0002                 cmp.b   #tt_DMA,d1              if DMA
1111    00000414 6710                      beq.s   di_isrdma
1112    00000416 08AA 0001 isr_end         bclr    #1,flags(a2)            if isr pending then do it
                0039
1113    0000041C 6706                      beq.s   di_isr_ex
1114    0000041E 4EB9 0000                 jsr     logint
                0000
1115    00000424 4E75      di_isr_ex       rts                             otherwise return
1116
```

```
1118                            *
1119                            * DMA transfer interrupt:
1120                            *
1121    00000426 286A 0040 di_isrdma       movea.l DMA_arm_addr(a2),a4     DMA arm & status register address
1122    0000042A 4A2B 000D                 tst.b   tdir_off(a3)            test direction
1123    0000042E 6632                      bne.s   isrdma_out              branch if output
1124
1125
1126                            *
1127                            * input DMA transfer
1128                            *
1129    00000430 41EB 0020                 lea     tfil_off(a3),a0         point to the buffer fill pointer
1130
1131    00000434 0829 0006                 btst    #6,status(a1)           last byte in tagged with EOI?
                 0017
1132    0000043A 56EA 0035                 sne     EOI_in(a2)              remember it
1133
1134    0000043E 082C 0000                 btst    #0,1(a4)                DMA channel unarmed?
                 0001
1135    00000444 670A                      beq.s   isrdma_unarmed          branch if so
1136
1137    00000446 0829 0001                 btst    #1,intreg(a1)           fifo idle?
                 0011
1138    0000044C 6608                      bne.s   isrdma_in_term          branch of so (terminate tfr)
1139    0000044E 60C6                      bra     isr_end                 otherwise, keep going
1140
1141    00000450 4AAA 003C isrdma_unarmed  tst.l   DMA_count(a2)           more to transfer?
1142    00000454 6E38                      bgt.s   isrdma_restart          branch if so
1143
1144    00000456 6100 021C isrdma_in_term  bsr     input_tfr_term          clean up the fifo's if possible
1145    0000045A 08A9 0001                 bclr    #1,intmask(a1)          re-mask fifo idle condition
                 0013
1146    00000460 6046                      bra.s   isrdma_term
1147
1148
1149                            *
1150                            * output DMA transfer
1151                            *
1152    00000462 41EB 001C isrdma_out      lea     temp_off(a3),a0         point to the buffer empty pointer
1153
1154    00000466 082C 0000                 btst    #0,1(a4)                DMA channel still armed?
                 0001
1155    0000046C 66A8                      bne.s   isr_end                 branch if so; keep going
1156
1157    0000046E 08E9 0007                 bset    #7,control(a1)          re-set 8-bit mode if necessary
                 0019
1158    00000474 4229 0017                 clr.b   status(a1)              clear the hi-order bits
1159
1160    00000478 0C6A 0001                 cmpi.w  #1,DMA_count(a2)        [ cmpi.l #$10000,DMA_count(a2) ]
                 003C
1161    0000047E 6D28                      blt.s   isrdma_term             branch if no more to transfer
1162
1163    00000480 6E0C                      bgt.s   isrdma_restart          branch if additional bursts remain
1164    00000482 4A2B 000B                 tst.b   tend_off(a3)            last burst: EOI tag set?
1165    00000486 6706                      beq.s   isrdma_restart          branch if not, otherwise...
1166    00000488 08A9 0007                 bclr    #7,control(a1)          clear 8-bit mode (tag last byte with EOI)
                 0019
```



```
1167
```

```
1169                        *
1170                        * DMA transfer restart
1171                        *
1172    0000048E 536A 003C isrdma_restart  subq.w  #1,DMA_count(a2)        [ sub.l #$10000,DMA_count(a2) ]
1173    00000492 202B 0010                 move.l  tcnt_off(a3),d0        previous burst length
1174    00000496 D190                      add.l   d0,(a0)                update the appropriate buffer pointer
1175    00000498 277C 0001                 move.l  #$10000,tcnt_off(a3)   next burst length
                 0000 0010
1176    000004A0 38AA 0044                 move    DMA_arm_word(a2),(a4)  re-arm the DMA channel
1177    000004A4 6000 FF70                 bra     isr_end                done!
1178
1179
1180                        *
1181                        * DMA transfer termination
1182                        *
1183    000004A8 137C 0080 isrdma_term     move.b  #$80,cardcontrol(a1)   disable card DMA
                 0003
1184    000004AE 2F08                      move.l  a0,-(sp)               save a0
1185    000004B0 4EB9 0000                 jsr     dropdma                free the DMA channel
                 0000
1186    000004B6 205F                      movea.l (sp)+,a0               restore a0
1187    000004B8 262B 0010                 move.l  tcnt_off(a3),d3        intended number of bytes transfered
1188    000004BC 9684                      sub.l   d4,d3                  actual number of bytes transferred
1189    000004BE D790                      add.l   d3,(a0)                update the appropriate buffer pointer
1190    000004C0 D8AA 003C                 add.l   DMA_count(a2),d4       total remaining count
1191    000004C4 2744 0010                 move.l  d4,tcnt_off(a3)        record it
1192    000004C8 4EB9 0000                 jsr     stclr                  mark the tfr done and log branch
                 0000
1193    000004CE 6000 FF46                 bra     isr_end                done!
1194
1195
1196
1197
1198                        *
1199                        * interrupt transfer processing:
1200                        *
1201    000004D2 4A2B 000D di_buf          tst.b   tdir_off(a3)           which direction transfer?
1202    000004D6 6602                      bne.s   di_bufo                branch if output
1203
1204
1205                        *
1206                        * buffered input:
1207                        *
1208    000004D8 4E71      di_bufi         nop
1209
1210                        *
1211                        * buffered output:
1212                        *
1213             0000 04DA di_bufo         equ *
1214
1215                        *
1216                        * int and frw transfer termination
1217                        *
1218             0000 04DA di_ti_term      equ *
1219
1220    000004DA 103C 0000 di_to_term      move.b  #0,d0                  disable other interrupts
1221                        *              bsr     di_eir
```



```
1222    000004DE 4EB9 0000                 jsr     stclr                  mark the buffer finished
                 0000
1223    000004E4 6000 FF30                 bra     isr_end                end of isr
1224
1225
1226                        *
1227                        * fast r/w transfer processing:
1228                        *
1229    000004E8 007C 2700 di_frw          ori     #$2700,sr                disable all other ints
1230                        *                                         the PASCAL system will re-enable&rte
1231    000004EC 4A2B 000D                 tst.b   tdir_off(a3)             which direction transfer?
1232    000004F0 6602                      bne.s   di_frwo                   skip if output
1233
1234                        *
1235                        * fast r/w input:
1236                        *
1237    000004F2 4E71      di_fri          nop
1238
1239                        *
1240                        * fast r/w output:
1241                        *
1242             0000 04F4 di_frwo         equ *
1243    000004F4 60E4                      bra     di_to_term             else we are done!
```

```
1245                        ********************************************************************************
1246                        *
1247                        *       isr1
1248                        *
1249                        *         the following routine does all the grunt work for the isr. it is
1250                        *         separated out so it can be called from background.
1251                        *
1252                        ********************************************************************************
1253
1254    000004F6 41E9 0011  di_isr1         lea     intreg(a1),a0           point to the interrupt condition register
1255    000004FA 49E9 0013                  lea     intmask(a1),a4          point to the interrupt mask register
1256                        *
1257                        * DCL
1258                        *
1259    000004FE 0810 0000                  btst    #0,(a0)                 DCL interrupt?
1260    00000502 6708                       beq.s   di_no_DCL               branch if not
1261    00000504 6100 0032                  bsr     di_log                  else do normal logging
1262    00000508 0894 0000                  bclr    #0,(a4)                 mask off the interrupt
1263             0000 050C  di_no_DCL       equ *
1264                        *
1265                        * SRQ
1266                        *
1267    0000050C 0810 0004                  btst    #4,(a0)                 SRQ interrupt?
1268    00000510 6708                       beq.s   di_no_SRQ               branch if not
1269    00000512 6100 0024                  bsr     di_log                  else do normal logging
1270    00000516 0894 0004                  bclr    #4,(a4)                 mask off the interrupt
1271             0000 051A  di_no_SRQ       equ *
1272                        *
1273                        * POL
1274                        *
1275    0000051A 0810 0005                  btst    #5,(a0)                 POL interrupt?
1276    0000051E 6708                       beq.s   di_no_POL               branch if not
1277    00000520 6100 0016                  bsr     di_log                  else do normal logging
1278    00000524 0894 0005                  bclr    #5,(a4)                 mask off the interrupt
1279             0000 0528  di_no_POL       equ *
1280                        *
1281                        * SCG
1282                        *
1283    00000528 0810 0007                  btst    #7,(a0)                 SCG interrupt?
1284    0000052C 6708                       beq.s   di_no_SCG               branch if not
1285    0000052E 6100 0008                  bsr     di_log                  else do normal logging
1286    00000532 0894 0007                  bclr    #7,(a4)                 mask off the interrupt
1287             0000 0536  di_no_SCG       equ *
1288    00000536 4E75                       rts
1289
1290                        *
1291                        * di_log    mark that an isr condition is pending
1292                        *
1293    00000538 08EA 0001  di_log          bset    #1,flags(a2)            set condition
                 0039
1294    0000053E 4E75                       rts
1295
```



```
1298                        ********************************************************************************
1299                        *
1300                        *       di_tfr
1301                        *
1302                        *         driver call for execution of tfr statement
1303                        *
1304                        *         entry:  conditions other than normal a1,a2 are:
1305                        *                 a3.l = pointer to transfer information
1306                        *
1307                        ********************************************************************************
1308
1309    00000540 4EB9 0000  di_tfr          jsr     check_tfr               wait for tfr to finish (timed)
                 0000
1310
1311    00000546 4A2B 000A                  tst.b   t_bw_off(a3)            don't allow word transfers
1312    0000054A 6600 FC92                  bne     di_noword
1313
1314    0000054E 202B 0010                  move.l  tcnt_off(a3),d0         get count
1315
1316    00000552 7200                       moveq   #0,d1
1317    00000554 122B 0009                  move.b  tusr_off(a3),d1         user specified transfer type
1318    00000558 D241                       add     d1,d1                   two bytes per table entry
1319    0000055A 4EB9 0000                  jsr     testdma                 DMA available?
                 0000
1320    00000560 6704                       beq.s   di_nodma                branch if not; use the first half of table
1321    00000562 D27C 0014                  add     #20,d1                  otherwise use the second half of table
1322    00000566 323B 1006  di_nodma        move    tfr_table(d1),d1        routine offset
1323    0000056A 4EFB 1002                  jmp     tfr_table(d1)           jump to the appropriate routine
1324
1325                        *
1326                        * transfer jump table
1327                        *
1328                        *                                                 --------------------    DMA uninstalled or unavailable
1329    0000056E FC68       tfr_table       dc      hterr_b-tfr_table       serial  interrupt
1330    00000570 FC6C                       dc      hterr_d-tfr_table       serial  DMA
1331    00000572 00D2                       dc      di_t_fhs-tfr_table      serial  fhs
1332    00000574 00D2                       dc      di_t_fhs-tfr_table      serial  fastest
1333    00000576 FC68                       dc      hterr_b-tfr_table       serial  overlap
1334                        *                                               --------------------
1335    00000578 00BE                       dc      di_t_int-tfr_table      overlap interrupt
1336    0000057A FC6C                       dc      hterr_d-tfr_table       overlap DMA
1337    0000057C 00C8                       dc      di_t_bst-tfr_table      overlap fhs
1338    0000057E 00C8                       dc      di_t_bst-tfr_table      overlap fastest
1339    00000580 00BE                       dc      di_t_int-tfr_table      overlap overlap
1340                        *                                               --------------------    DMA available
1341    00000582 FC68                       dc      hterr_b-tfr_table       serial  interrupt
1342    00000584 0028                       dc      di_t_DMA-tfr_table      serial  DMA
1343    00000586 00D2                       dc      di_t_fhs-tfr_table      serial  fhs
1344    00000588 0028                       dc      di_t_DMA-tfr_table      serial  fastest
1345    0000058A FC68                       dc      hterr_b-tfr_table       serial  overlap
1346                        *                                               --------------------
1347    0000058C 00BE                       dc      di_t_int-tfr_table      overlap interrupt
1348    0000058E 0028                       dc      di_t_DMA-tfr_table      overlap DMA
1349    00000590 00C8                       dc      di_t_bst-tfr_table      overlap fhs
1350    00000592 0028                       dc      di_t_DMA-tfr_table      overlap fastest
1351    00000594 0028                       dc      di_t_DMA-tfr_table      overlap overlap
```

```
1353                            *
1354                            * transfer DMA
1355                            *
1356    00000596 177C 0002 di_t_DMA         move.b  #tt_DMA,tact_off(a3)    set tfr type to DMA
                 0007
1357
1358    0000059C 2200                       move.l  d0,d1                   total count
1359    0000059E 5381                       subq.l  #1,d1                   total count minus 1
1360    000005A0 7000                       moveq   #0,d0
1361    000005A2 3001                       move.w  d1,d0                   first burst count minus 1
1362    000005A4 9280                       sub.l   d0,d1                   remaining count (multiple of $10000)
1363    000005A6 2541 003C                  move.l  d1,DMA_count(a2)        remember it
1364    000005AA 5280                       addq.l  #1,d0                   burst count [1..65536]
1365
1366    000005AC 4229 0003                  clr.b   cardcontrol(a1)         disable card interrupts
1367    000005B0 4EB9 0000                  jsr     getdma                  get a DMA channel
                 0000
1368    000005B6 0882 0000                  bclr    #0,d2                   always clear the DMA interrupt enable bit
1369    000005BA 4A2B 0030                  tst.b   t_DMApri(a3)            DMA priority requested?
1370    000005BE 6704                       beq.s   di_arm_DMA              branch if not
1371    000005C0 08C2 0003                  bset    #3,d2                   otherwise set the DMA priority bit
1372    000005C4 3882       di_arm_DMA      move    d2,(a4)                 arm the channel
1373    000005C6 254C 0040                  move.l  a4,DMA_arm_addr(a2)     remember the DMA channel arm address
1374    000005CA 3542 0044                  move    d2,DMA_arm_word(a2)     remember the DMA channel arm word
1375
1376    000005CE 49F8 0000                  lea     0,a4                    no DMA termination routine (no DMA interrupt enabled)
1377    000005D2 4EB9 0000                  jsr     DMA_stbsy               set DMA channel, buffer busy
                 0000
1378
1379    000005D8 4A2B 000D                  tst.b   tdir_off(a3)            test the transfer direction
1380    000005DC 6614                       bne.s   di_tod                  branch if output
1381
1382                            *
1383                            * transfer input DMA:
1384                            *
1385    000005DE 08E9 0001 di_tid           bset    #1,intmask(a1)          interrupt on DMA completion or fifo idle
                 0013
1386    000005E4 08A9 0001                  bclr    #1,control(a1)          select the inbound fifo for DMA
                 0019
1387    000005EA D081                       add.l   d1,d0                   recompute the total count again
1388    000005EC 6100 005C                  bsr     input_tfr_en            enable the transfer
1389    000005F0 6024                       bra.s   di_DMA_en               enable the card & wait if serial
1390
1391
1392                            *
1393                            * transfer output DMA:
1394                            *
1395    000005F2 08E9 0001 di_tod           bset    #1,control(a1)          select the outbound fifo for DMA
                 0019
1396    000005F8 08E9 0007                  bset    #7,control(a1)          assume no EOI tag (8-bit mode)
                 0019
1397    000005FE 4229 0017                  clr.b   status(a1)              clear the high-order bits
1398    00000602 4A81                       tst.l   d1                      remaining count 0?
1399    00000604 660C                       bne.s   di_tod_1                branch if not
1400    00000606 4A2B 000B                  tst.b   tend_off(a3)            EOI tag set?
1401    0000060A 6706                       beq.s   di_tod_1                branch if not, otherwise...
```



```
1402    0000060C 08A9 0007                  bclr    #7,control(a1)          clear 8-bit mode (tag last byte with EOI)
                 0019
1403    00000612 08C3 0003 di_tod_1         bset    #3,d3                   set the card's output DMA enable bit
1404    00000616 1343 0003 di_DMA_en        move.b  d3,cardcontrol(a1)      enable card interrupts & DMA
1405
1406    0000061A 0C2B 0005 di_DMA_w         cmpi.b  #5,tusr_off(a3)         overlap transfer?
                 0009
1407    00000620 6C08                       bge.s   di_DMA_w2               branch if so
1408
1409    00000622 0C2B 00FF di_DMA_w1        cmpi.b  #255,t_sc_off(a3)       if not then wait till done
                 0005
1410    00000628 66F8                       bne.s   di_DMA_w1
1411
1412    0000062A 4E75       di_DMA_w2       rts
1413
```

```
1415                        *
1416                        * transfer interrupt
1417                        *
1418    0000062C 177C 0001 di_t_int        move.b  #tt_int,tact_off(a3)    set tfr type to interrupt
                0007
1419    00000632 6000 FBA2                 bra     hterr_b
1420
1421
1422                        *
1423                        * transfer burst (intr on 1st byte fhs on rest)
1424                        *
1425    00000636 177C 0003 di_t_bst        move.b  #tt_burst,tact_off(a3)  set tfr type to burst
                0007
1426    0000063C 6000 FB98                 bra     hterr_b
1427
1428
1429                        *
1430                        * transfer fast handshake
1431                        *
1432    00000640 177C 0004 di_t_fhs        move.b  #tt_fhs,tact_off(a3)    set tfr type to fhs
                0007
1433    00000646 6000 FB8E                 bra     hterr_b
1434
```



```
1436                        *
1437                        * input transfer enable
1438                        *
1439    0000064A 0829 0004 input_tfr_en    btst    #4,status(a1)           active controller?
                0017
1440    00000650 6714                      beq.s   te_rts                  branch if not
1441
1442    00000652 B0BC 0000                 cmp.l   #256,d0                 count low enough for counted transfer?
                0100
1443    00000658 6F0E                      ble.s   te_ect                  branch if so
1444
1445    0000065A 137C 00C0                 move.b  #$C0,status(a1)
                0017
1446    00000660 137C 0000                 move.b  #$00,fifo(a1)           uncounted transfer enable
                0015
1447    00000666 4E75       te_rts         rts
1448
1449    00000668 137C 0080 te_ect          move.b  #$80,status(a1)         inhibit LF detection
                0017
1450    0000066E 1340 0015                 move.b  d0,fifo(a1)             counted transfer enable
1451    00000672 4E75                      rts
1452
1453
1454                        *
1455                        * input transfer term
1456                        *
1457    00000674 0829 0004 input_tfr_term  btst    #4,status(a1)           active controller?
                0017
1458    0000067A 6744                      beq.s   tt_rts                  branch if not
1459
1460    0000067C 0829 0001                 btst    #1,intreg(a1)           fifo idle?
                0011
1461    00000682 6622                      bne.s   tt_empty_fifo           branch if so
1462                        *
1463                        * (input) transfer enable byte still active - must init outbound fifo
1464                        *
1465    00000684 0829 0003                 btst    #3,status(a1)           system controller?
                0017
1466    0000068A 6714                      beq.s   tt_non_sys              branch if not
1467
1468    0000068C 0029 0011                 ori.b    #$11,control(a1)       assert IFC & init outbound fifo
                0019
1469    00000692 7050                      moveq   #80,d0                  must hold IFC at least 100us...
1470    00000694 51C8 FFFE                 dbra    d0,*                    to satisfy the bus standard
1471    00000698 08A9 0004                 bclr    #4,control(a1)          release IFC
                0019
1472    0000069E 6006                      bra.s   tt_empty_fifo           go empty the inbound fifo
1473
1474    000006A0 08E9 0000 tt_non_sys      bset    #0,control(a1)          init outbound fifo; TCA like this is hazardous!!!
                0019
1475                        *
1476                        * empty inbound fifo
1477                        *
1478    000006A6 08E9 0002 tt_empty_fifo   bset    #2,intmask(a1)          unmask fifo byte condition
                0013
1479    000006AC 0829 0002                 btst    #2,intreg(a1)           fifo byte?
                0011
```

```
1480    000006B2 6706                       beq.s   tt_fifo_empty           branch if not
1481    000006B4 4A29 0015                  tst.b   fifo(a1)                otherwise, discard it and...
1482    000006B8 60EC                       bra     tt_empty_fifo           go check for more
1483
1484    000006BA 08A9 0002 tt_fifo_empty    bclr    #2,intmask(a1)          re-mask fifo byte condition
                 0013
1485
1486    000006C0 4E75      tt_rts           rts
1487
1488
1489                                        end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ABORT_IO	ABS	130	00000015
CHECK_TFR	ABS	132	0000001B
DMA_STBSY	ABS	128	00000010
DROPDMA	ABS	121	00000002
GETDMA	ABS	122	00000004
IODECLARATIONS	ABS	275	0000001E
ITXFR	ABS	129	00000013
LOGEOT	ABS	125	0000000A
LOGINT	ABS	124	00000008
STBSY	ABS	126	0000000C
STCLR	ABS	127	0000000E
SYSGLOBALS	ABS	276	00000022
TESTDMA	ABS	123	00000006
WAIT_TFR	ABS	131	00000018

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ABI	ABS	321		00000034
ADDRESS	ABS	549		0000001B
AVAIL_OFF	ABS	215		00000034
BADTMO	ABS	293		0000000B
BAD_RDS	ABS	301		00000013
BAD_SCT	ABS	302		00000014
BUFI_OFF	ABS	206		00000024
BUFO_OFF	ABS	207		00000028
BUF_BUSY	ABS	291		00000009
CARDCONTROL	ABS	541		00000003
CARDID	ABS	538		00000001
CARDLATCH	ABS	542		00000007
CARDRESET	ABS	539		00000001
CARDSTATUS	ABS	540		00000003
CCR	STREG	0		00000005
CLEAR_TABLE	REL	1018		000003A8
CONTROL	ABS	548		00000019
CRD_DWN	ABS	303		00000015
C_ADR	ABS	205		00000020
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005

```
D6              DREG       0                          00000006
D7              DREG       0                          00000007
DCL             ABS      566                          00000014
DI_ARM_DMA      REL     1372                          000005C4
DI_BUF          REL     1201                          000004D2
DI_BUFI         REL     1208                          000004D8
DI_BUFO         REL     1213                          000004DA
DI_CLR          REL     1012                          00000396
DI_DMA_EN       REL     1404                          00000616
DI_DMA_W        REL     1406                          0000061A
DI_DMA_W1       REL     1409                          00000622
DI_DMA_W2       REL     1412                          0000062A
DI_DMYRTS       REL     1034                          000003C8
DI_EOI          REL      998                          00000390
DI_FRI          REL     1237                          000004F2
DI_FRW          REL     1229                          000004E8
DI_FRWO         REL     1242                          000004F4
DI_IFC          REL      981                          0000036A
DI_INIT         REL      584                          0000011E
DI_INIT_S       REL      602                          00000128
DI_ISR          REL     1102                          000003EE
DI_ISR1         REL     1254                          000004F6
DI_ISRDMA       REL     1121                          00000426
DI_ISR_EX       REL     1115                          00000424
DI_LOCAL        REL     1031                          000003B8
DI_LOG          REL     1293                          00000538
DI_LSTERR       REL      705                          000001E2
DI_NODMA        REL     1322                          00000566
DI_NOTACTL      REL      690                          000001CE
DI_NOTSCTL      REL      693                          000001D2
DI_NOWORD       REL      702                          000001DE
DI_NO_DCL       REL     1263                          0000050C
DI_NO_POL       REL     1279                          00000528
DI_NO_SCG       REL     1287                          00000536
DI_NO_SRQ       REL     1271                          0000051A
DI_P_POLL       REL      926                          00000300
DI_RDOUT        REL     1075                          000003CE
DI_RDB          REL      638                          0000017A
DI_RDB1         REL      645                          00000194
DI_RDS          REL      783                          0000024A
DI_REN          REL      972                          00000358
DI_RQS          REL      893                          000002D0
DI_RQS2         REL      899                          000002E4
DI_SCBSY        REL      684                          000001C6
DI_SC_ERR       REL      687                          000001CA
DI_SET          REL      953                          00000336
DI_TEST         REL     1057                          000003CA
DI_TFR          REL     1309                          00000540
DI_TID          REL     1385                          000005DE
DI_TI_TERM      REL     1218                          000004DA
DI_TLKERR       REL      708                          000001E6
DI_TMO          REL      711                          000001EA
DI_TOD          REL     1395                          000005F2
DI_TOD_1        REL     1403                          00000612
DI_TO_TERM      REL     1220                          000004DA
DI_T_BST        REL     1425                          00000636
DI_T_DMA        REL     1356                          00000596
```



```
DI_T_FHS        REL     1432                          00000640
DI_T_INT        REL     1418                          0000062C
DI_WAIT_FB      REL      731                          00000204
DI_WAIT_FI      REL      728                          00000200
DI_WFC          REL      736                          00000206
DI_WFC_DONE     REL      769                          0000023E
DI_WFC_INFINITE REL      764                          0000023A
DI_WFC_QUICK    REL      743                          00000214
DI_WFC_TIMED    REL      753                          00000226
DI_WTB          REL      666                          000001A8
DI_WTC          REL      863                          000002AA
DI_WTC_RST      REL      880                          000002BA
DMA_ARM_ADDR    ABS      341                          00000040
DMA_ARM_WORD    ABS      342                          00000044
DMA_COUNT       ABS      340                          0000003C
EIRB_OFF        ABS      208                          0000002C
EOD_SEEN        ABS      304                          00000016
EOI_IN          ABS      322                          00000035
EOI_OUT         ABS      323                          00000036
ESC_CODE        ABS      312 SYSGLOBALS      +        FFFFFFFE
ESC_ERR         REL      715                          000001EC
EXTDI_EDI_CLR   REL      496                          000000E2
EXTDI_EDI_END   REL      520                          00000108
EXTDI_EDI_INIT  REL      360                          00000002
EXTDI_EDI_ISR   REL      369                          00000010
EXTDI_EDI_PPOLL REL      474                          000000BE
EXTDI_EDI_RDB   REL      378                          0000001E
EXTDI_EDI_RDS   REL      429                          00000076
EXTDI_EDI_RDW   REL      401                          00000042
EXTDI_EDI_SEND  REL      464                          000000AE
EXTDI_EDI_SET   REL      486                          000000D2
EXTDI_EDI_TEST  REL      507                          000000F2
EXTDI_EDI_TFR   REL      454                          0000009E
EXTDI_EDI_WTB   REL      390                          00000032
EXTDI_EDI_WTC   REL      442                          0000008C
EXTDI_EDI_WTW   REL      415                          0000005C
EXTDI_EXTDI     REL      355                          00000000
FIFO            ABS      546                          00000015
FLAGS           ABS      325                          00000039
GET             ABS      563                          00000008
GTL             ABS      560                          00000001
HPL_WTC1        REL      882                          000002BE
HPL_WTC2        REL      887                          000002CE
HPL_WTC4        REL      901                          000002EA
HPL_WTC5        REL      907                          000002FA
HTERR_B         REL      696                          000001D6
HTERR_D         REL      699                          000001DA
HWTCTBL         REL      869                          000002B6
H_ISR_PM        ABS      204                          0000001C
H_ISR_PR        ABS      202                          00000014
H_ISR_SL        ABS      203                          00000018
INPUT_TFR_EN    REL     1439                          0000064A
INPUT_TFR_TERM  REL     1457                          00000674
INTMASK         ABS      545                          00000013
INTREG          ABS      544                          00000011
IOE_ERROR       ABS      307                          FFFFFFE6
IOE_RSLT        ABS      309 IODECLARATIONS  +        FFFFFFBE
```

IOE_SC	ABS	310	IODECLARATIONS	+	FFFFFFBA
IO_MISC	ABS	305			00000017
IO_SC	ABS	209			0000002D
ISRDMA_IN_TERM	REL	1144			00000456
ISRDMA_OUT	REL	1152			00000462
ISRDMA_RESTART	REL	1172			0000048E
ISRDMA_TERM	REL	1183			000004A8
ISRDMA_UNARMED	REL	1141			00000450
ISR_END	REL	1112			00000416
ISR_ENTRY	ABS	200			00000000
LLO	ABS	565			00000011
MA	ABS	214			00000033
MA_W	ABS	213			00000032
NOT_HPIB	ABS	284			00000002
NOT_LSTN	ABS	298			00000010
NOT_TALK	ABS	297			0000000F
NO_ACTL	ABS	285			00000003
NO_CARD	ABS	283			00000001
NO_DATA	ABS	288			00000006
NO_DMA	ABS	295			0000000D
NO_DRV	ABS	294			0000000C
NO_DVC	ABS	286			00000004
NO_SCTL	ABS	300			00000012
NO_SPACE	ABS	287			00000005
NO_WORD	ABS	296			0000000E
PPC	ABS	562			00000005
PPD	ABS	573			00000070
PPE	ABS	572			00000060
PPOLLMSK	ABS	337			0000003A
PPOLMASK	ABS	550			0000001D
PPOLSENSE	ABS	551			0000001F
PPU	ABS	567			00000015
RCVR_BLK	ABS	313	SYSGLOBALS	+	FFFFFFF6
RDS_0	REL	816			00000272
RDS_1	REL	802			0000026C
RDS_2	REL	803			0000026C
RDS_3	REL	823			00000276
RDS_4	REL	804			0000026C
RDS_5	REL	805			0000026C
RDS_6	REL	836			0000028A
RDS_7	REL	806			0000026C
RDS_8	REL	807			0000026C
RDS_ERR	REL	809			0000026C
RDS_TABLE	REL	792			0000025A
SC_BUSY	ABS	290			00000008
SDC	ABS	561			00000004
SET_TABLE	REL	959			00000348
SP	AREG	0			00000007
SPD	ABS	569			00000019
SPE	ABS	568			00000018
SR	STREG	0			00000006
STATUS	ABS	547			00000017
TACT_OFF	ABS	230			00000007
TBSZ_OFF	ABS	253			00000018
TBUF_OFF	ABS	252			00000014
TCHR_OFF	ABS	249			0000000E
TCNTERR	ABS	292			0000000A

TCNT_OFF	ABS	251		00000010
TCT	ABS	564		00000009
TDIR_OFF	ABS	247		0000000D
TEMP_OFF	ABS	254		0000001C
TEND_OFF	ABS	245		0000000B
TE_ECT	REL	1449		00000668
TE_RTS	REL	1447		00000666
TFIL_OFF	ABS	255		00000020
TFR_ERR	ABS	289		00000007
TFR_TABLE	REL	1329		0000056E
TIMEOUT	ABS	210		0000002E
TMO_ERR	ABS	299		00000011
TTMP_OFF	ABS	228		00000000
TT_BURST	ABS	266		00000003
TT_DMA	ABS	265		00000002
TT_EMPTY_FIFO	REL	1478		000006A6
TT_FHS	ABS	267		00000004
TT_FIFO_EMPTY	REL	1484		000006BA
TT_INT	ABS	264		00000001
TT_NON_SYS	REL	1474		000006A0
TT_RTS	REL	1486		000006C0
TUSR_OFF	ABS	231		00000009
T_BW_OFF	ABS	243		0000000A
T_DMAPRI	ABS	260		00000030
T_PM_OFF	ABS	259		0000002C
T_PR_OFF	ABS	256		00000024
T_SC_OFF	ABS	229		00000005
T_SL_OFF	ABS	258		00000028
UNL	ABS	570		0000003F
UNT	ABS	571		0000005F
USER_ISR	ABS	201		00000014
USP	STREG	0		00000007

DRVASM

Purpose

DRVASM contains common assembly language routines used by the DISCHPIB, AMIGO and CS80 modules.

```
 PAGE   1 [2.0] 11/4/82 13:50:19  ASSEMBLY OF DRVASM.TEXT
PASS 1 COMPLETE. ERRORS: 0
    1                                       nosyms
    2
    3                       ********************************************************************************
    4                       *                          driver assembly routines                            *
    5                       ********************************************************************************
    6
    7                       *
    8                       * PASCAL interface text
    9                       *
   10                                       mname   drvasm
   11
   12                        src    module drvasm;
   13                        src
   14                        src      import
   15                        src        sysglobals;
   16                        src
   17                        src      export
   18                        src        function test_and_toggle(var semaphore: boolean): boolean;
   19                        src        procedure eor(correction_byte: char; bufptr: charptr);
   20                        src
   21                        src    end; {drvasm}
   22
   23
   24                      *
   25                      * def's
   26                      *
   27                                       def     drvasm_drvasm
   28                                       def     drvasm_test_and_toggle
   29                                       def     drvasm_eor
   30
   31
   32                      *
   33                      * module intialization routine
   34                      *
   35    00000000 4E75     drvasm_drvasm   rts
   36
   37
   38                      *
   39                      * test and toggle  -  semaphore manipulation function
   40                      *
   41            0000 0002 drvasm_test_and_toggle equ *
   42    00000002 205F                     movea.l (sp)+,a0        pop the return address
   43    00000004 225F                     movea.l (sp)+,a1        pop the var parameter address
   44    00000006 0851 0000                bchg    #0,(a1)         test and toggle the semaphore
   45    0000000A 56C0                     sne     d0              remember the previous state
   46    0000000C 4400                     neg.b   d0              form a legal PASCAL boolean
   47    0000000E 1E80                     move.b  d0,(sp)         set the return variable
   48    00000010 4ED0                     jmp     (a0)            return
   49
   50
   51                      *
   52                      * exclusive or  -  error correction procedure
   53                      *
   54    00000012 205F     drvasm_eor      movea.l (sp)+,a0        pop the return address
   55    00000014 225F                     movea.l (sp)+,a1        pop the bufptr
   56    00000016 101F                     move.b  (sp)+,d0        pop the correction character
   57    00000018 B111                     eor.b   d0,(a1)         do it to it
   58    0000001A 4ED0                     jmp     (a0)            return


 PAGE   2 [2.0] 11/4/82 13:50:19  ASSEMBLY OF DRVASM.TEXT

   59
   60                                       end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

EVALGVR

Purpose

EVALGVR contains code to quickly evaluate a general value record (GVR) and to assist in relocating object code.

Usage

EVALGVR is used by the loader.

```
PASS 1 COMPLETE. ERRORS: 0
    1    00000000                    rorg 0
    2                                nosyms
    3
    4                                def     evalgvr,relocate
    5                                refa    sysglobals,loader
    6
    7             FFFF FFFE escapecode       equ sysglobals-2          globals:
    8             FFFF FFE4 newmods          equ loader-28
    9
   10             0000 0006 defaddr          equ 6             fields of mdb
   11             0000 000A defsize          equ 10
   12             0000 0010 extaddr          equ 16
   13             0000 0026 relocdelta       equ 38
   14             0000 002A globaldelta      equ 42
   15
   16             0000 0001 op               equ 1             fields in gvr
   17             0000 0000 done             equ 0
   18
   19             0000 0008 longoffset       equ 8             fields in a gvr:
   20             0000 0009 valueextend      equ 9
   21             0000 000A patchable        equ 10
   22             0000 000B datasize         equ 11
   23             0000 000E primarytype      equ 14
   24
   25             0000 0064 sbyteerr         equ 100           arithmetic errors
   26             0000 0065 sworderr         equ 101
   27             0000 0066 sinterr          equ 102
   28             0000 0068 ubyteerr         equ 104
   29             0000 0069 uworderr         equ 105
   30
   31             0000 006E deptherr         equ 110
   32             0000 006F formaterr        equ 111
   33
   34             0000 000A maxdepth         equ 10
   35
   36             0000 0000 gvalue  equ d0
   37             0000 0001 sub     equ d1
   38             0000 0002 gvr     equ d2
   39             0000 0003 field   equ d3
   40             0000 0004 dtemp   equ d4
   41             0000 0005 depth   equ d5
   42
   43             0000 0000 gvrptr  equ a0
   44             0000 0001 modptr  equ a1
   45                       *above used by runlist
   46             0000 0006 aobject equ d6
   47             0000 0002 atemp   equ a2
   48             0000 0003 return  equ a3
   49             0000 0004 object  equ a4
   50
   51                       *proc relocate(refbuftop, refindex: address;
   52                       *              var object:  address;
   53                       *              modptr:  moddescptr);
   54
   55    00000000 0050      jtable  dc.w    sbyte-jtable
   56    00000002 0068              dc.w    sword-jtable
   57    00000004 007E              dc.w    sint-jtable
   58    00000006 0010              dc.w    ferror-jtable   reserved
```



```
   59    00000008 0086              dc.w    ubyte-jtable
   60    0000000A 009A              dc.w    uword-jtable
   61    0000000C 0010              dc.w    ferror-jtable   reserved
   62    0000000E 0010              dc.w    ferror-jtable   reserved
   63
   64    00000010 3B7C 006F ferror  move.w  #formaterr,escapecode(a5)
                  FFFE
   65    00000016 4E4A              trap    #10
   66
   67    00000018 265F      relocate movea.l (sp)+,return
   68    0000001A 225F               movea.l (sp)+,modptr
   69    0000001C 285F               movea.l (sp)+,object
   70    0000001E 2C14               move.l  (object),aobject
   71    00000020 CD8C               exg     object,aobject
   72    00000022 205F               movea.l (sp)+,gvrptr    refindex
   73    00000024 245F               movea.l (sp)+,atemp     refbuftop
   74    00000026 4201               clr.b   sub
   75    00000028 7A0A               moveq   #maxdepth,depth
   76    0000002A 6000 0086          bra     nextref
   77
   78    0000002E 3418      refloop move.w  (gvrptr)+,gvr   get ref record
   79    00000030 4283               clr.l   field
   80    00000032 1602               move.b  gvr,field       short offset
   81    00000034 0802 0008          btst    #longoffset,gvr
   82    00000038 6704               beq.s   short
   83    0000003A 4843               swap    field           shift left 16
   84    0000003C 3618               move.w  (gvrptr)+,field
   85    0000003E D9C3      short    adda.l  field,object
   86    00000040 3602               move    gvr,field       get data size
   87    00000042 ED5B               rol.w   #16-datasize+1,field
   88    00000044 C67C 000E          and     #7+7,field      3 bits
   89    00000048 383B 30B6          move.w  jtable(field.w),dtemp
   90    0000004C 4EFB 40B2          jmp     jtable(dtemp.w)
   91
   92    00000050 1014      sbyte    move.b  (object),gvalue
   93    00000052 4880               ext.w   gvalue
   94    00000054 48C0               ext.l   gvalue
   95    00000056 6164               bsr.s   runref
   96    00000058 1880               move.b  gvalue,(object)
   97    0000005A 7818               moveq   #32-8,dtemp
   98    0000005C E9A0               asl.l   dtemp,gvalue
   99    0000005E 6852               bvc.s   nextref
  100    00000060 3B7C 0064          move.w  #sbyteerr,escapecode(a5)
                  FFFE
  101    00000066 6068               bra.s   linkerr
  102
  103    00000068 3014      sword    move.w  (object),gvalue
  104    0000006A 48C0               ext.l   gvalue
  105    0000006C 614E               bsr.s   runref
  106    0000006E 3880               move.w  gvalue,(object)
  107    00000070 7810               moveq   #32-16,dtemp
  108    00000072 E9A0               asl.l   dtemp,gvalue
  109    00000074 683C               bvc.s   nextref
  110    00000076 3B7C 0065          move.w  #sworderr,escapecode(a5)
                  FFFE
  111    0000007C 6052               bra.s   linkerr
  112
```

```
 113    0000007E 2014       sint    move.l  (object),gvalue
 114    00000080 613A               bsr.s   runref
 115    00000082 2880               move.l  gvalue,(object)
 116    00000084 602C               bra.s   nextref
 117
 118    00000086 4280       ubyte   clr.l   gvalue
 119    00000088 1014               move.b  (object),gvalue
 120    0000008A 6130               bsr.s   runref
 121    0000008C 1880               move.b  gvalue,(object)
 122    0000008E E080               asr.l   #8,gvalue
 123    00000090 6720               beq.s   nextref
 124    00000092 3B7C 0068          move.w  #ubyteerr,escapecode(a5)
                 FFFE
 125    00000098 6036               bra.s   linkerr
 126
 127    0000009A 4280       uword   clr.l   gvalue
 128    0000009C 3014               move.w  (object),gvalue
 129    0000009E 611C               bsr.s   runref
 130    000000A0 3880               move.w  gvalue,(object)
 131    000000A2 C0BC FFFF          and.l   #$FFFF0000,gvalue
                 0000
 132    000000A8 6708               beq.s   nextref
 133    000000AA 3B7C 0069          move.w  #uworderr,escapecode(a5)
                 FFFE
 134    000000B0 601E               bra.s   linkerr
 135
 136    000000B2 B1CA       nextref cmpa.l  atemp,gvrptr
 137    000000B4 6D00 FF78          blt     refloop
 138
 139    000000B8 4ED3               jmp     (return)
 140
 141    000000BA 3418       runlist move.w  (gvrptr)+,gvr   get gvr record
 142    000000BC 0802 0009  runref  btst    #valueextend,gvr
 143    000000C0 6718               beq.s   pt
 144    000000C2 4A01               tst.b   sub
 145    000000C4 6A10               bpl.s   addit
 146    000000C6 9098               sub.l   (gvrptr)+,gvalue
 147    000000C8 6810               bvc.s   pt
 148    000000CA 3B7C 0066  intover move.w  #sinterr,escapecode(a5)
                 FFFE
 149    000000D0 CD8C       linkerr exg     object,aobject
 150    000000D2 2886               move.l  aobject,(object)
 151    000000D4 4E4A               trap    #10
 152    000000D6 D098       addit   add.l   (gvrptr)+,gvalue
 153    000000D8 69F0               bvs     intover
 154    000000DA E55A       pt      rol.w   #16-primarytype,gvr     type
 155    000000DC C47C 0003          and     #3,gvr                  2 bits
 156    000000E0 677C               beq.s   endrun          done if abs
 157    000000E2 2809               move.l  modptr,dtemp    find module
 158    000000E4 6618               bne.s   notnil
 159    000000E6 226D FFE4          movea.l newmods(a5),modptr
 160    000000EA 6002               bra.s   mstart
 161    000000EC 2251       mloop   movea.l (modptr),modptr         link
 162    000000EE 2829 0006  mstart  move.l  defaddr(modptr),dtemp
 163    000000F2 B1C4               cmpa.l  dtemp,gvrptr
 164    000000F4 6FF6               ble.s   mloop
 165    000000F6 D8A9 000A          add.l   defsize(modptr),dtemp
```



```
 166    000000FA B1C4               cmpa.l  dtemp,gvrptr
 167    000000FC 6EEE               bgt.s   mloop
 168    000000FE 5542       notnil  subq    #2,gvr          primary type
 169    00000100 6F5E               ble.s   addref          reloc or global
 170    00000102 51CD 000A          dbra    depth,rloop     limit recursion
 171    00000106 3B7C 006E          move.w  #deptherr,escapecode(a5)
                 FFFE
 172    0000010C 4E4A               trap    #10
 173    0000010E 3418       rloop   move.w  (gvrptr)+,gvr
 174    00000110 3801               move    sub,dtemp       save for later
 175    00000112 0802 0001          btst    #op,gvr
 176    00000116 6702               beq.s   nosub
 177    00000118 4601               not.b   sub
 178    0000011A 3602       nosub   move    gvr,field
 179    0000011C C67C FFFC          and     #$FFFC,field    adr field
 180    00000120 B67C 0004          cmp     #4,field
 181    00000124 6E06               bgt.s   defref
 182    00000126 6138               bsr.s   addref
 183    00000128 3204               move    dtemp,sub
 184    0000012A 602A               bra.s   nextr
 185    0000012C 2F09       defref  move.l  modptr,-(sp)    save modptr
 186    0000012E 2F08               move.l  gvrptr,-(sp)    save gvrptr
 187    00000130 3F02               move.w  gvr,-(sp)       save done
 188    00000132 1F04               move.b  dtemp,-(sp)     save sub
 189    00000134 2069 0010          movea.l extaddr(modptr),gvrptr
 190    00000138 2070 3000          movea.l 0(gvrptr,field.w),gvrptr
 191    0000013C 4244               clr     dtemp
 192    0000013E 3244               movea.w dtemp,modptr    modptr := NIL
 193    00000140 1810               move.b  (gvrptr),dtemp  length of symbol
 194    00000142 5444               addq    #2,dtemp        skip symbol
 195    00000144 0884 0000          bclr    #0,dtemp
 196    00000148 D0C4               adda.w  dtemp,gvrptr
 197    0000014A 6100 FF6E          bsr     runlist
 198    0000014E 121F               move.b  (sp)+,sub       restore sub
 199    00000150 341F               move.w  (sp)+,gvr       restore done
 200    00000152 205F               movea.l (sp)+,gvrptr    restore gvrptr
 201    00000154 225F               movea.l (sp)+,modptr    save modptr
 202    00000156 0802 0000  nextr   btst    #done,gvr
 203    0000015A 67B2               beq     rloop
 204    0000015C 5245               addq    #1,depth
 205    0000015E 4E75       endrun  rts
 206
 207    00000160 6D18       addref  blt.s   reloc
 208
 209    00000162 4A01       global  tst.b   sub
 210    00000164 6A0A               bpl.s   addglobal
 211    00000166 90A9 002A          sub.l   globaldelta(modptr),gvalue
 212    0000016A 6900 FF5E          bvs     intover
 213    0000016E 4E75               rts
 214    00000170 D0A9 002A  addglobal add.l globaldelta(modptr),gvalue
 215    00000174 6900 FF54          bvs     intover
 216    00000178 4E75               rts
 217
 218    0000017A 4A01       reloc   tst.b   sub
 219    0000017C 6A0A               bpl.s   addreloc
 220    0000017E 90A9 0026          sub.l   relocdelta(modptr),gvalue
 221    00000182 6900 FF46          bvs     intover
```

```
 PAGE   5 [2.0] 11/4/82 13:14:25  ASSEMBLY OF EVALGVR.TEXT

  222    00000186 4E75               rts
  223    00000188 D0A9 0026 addreloc add.l  relocdelta(modptr),gvalue
  224    0000018C 6900 FF3C          bvs    intover
  225    00000190 4E75               rts
  226
  227                       *proc evalgvr(var gvalue, gvrptr:  address;
  228                       *                 modptr: moddescptr);
  229
  230    00000192 265F      evalgvr movea.l (sp)+,return    return address
  231    00000194 225F              movea.l (sp)+,modptr
  232    00000196 245F              movea.l (sp)+,atemp     var gvrptr
  233    00000198 2052              movea.l (atemp),gvrptr
  234    0000019A 4280              clr.l   gvalue
  235    0000019C 4201              clr.b   sub
  236    0000019E 7A0A              moveq   #maxdepth,depth        allow nesting
  237    000001A0 6100 FF18         bsr     runlist
  238    000001A4 2488              move.l  gvrptr,(atemp)
  239    000001A6 245F              movea.l (sp)+,atemp     var gvalue
  240    000001A8 2480              move.l  gvalue,(atemp)
  241    000001AA 4ED3              jmp     (return)
  242
  243
  244                               end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

FASTMOVE

Purpose

FASTMOVE implements the data moving procedures ASM_MOVELEFT and ASM_MOVERIGHT as well as a similar procedure called FASTMOVE.

Notes

FASTMOVE determines which direction the bytes should be moved. MOVELEFT and MOVERIGHT may have unpredictable results if the direction is opposite that which is appropriate.

```
PASS 1 COMPLETE. ERRORS: 0
    1                           *       UNIFIED GENERAL PURPOSE BYTE MOVE OPERATION
    2
    3                                   def     asm_fastmove,asm_movel,asm_mover,asm_moveleft,asm_moveright
    4
    5               0000 0000 length            equ     d0
    6               0000 0001 delta             equ     d1
    7
    8               0000 0000 return            equ     a0
    9               0000 0001 source            equ     a1
   10               0000 0002 destination       equ     a2
   11
   12
   13
   14               0000 0000 asm_movel         equ *
   15               0000 0000 asm_moveleft      equ *
   16      00000000 205F                movea.l (sp)+,return
   17      00000002 201F                move.l  (sp)+,length
   18      00000004 6F18                ble.s   nonpos                  length must be > 0
   19      00000006 245F                movea.l (sp)+,destination
   20      00000008 225F                movea.l (sp)+,source
   21      0000000A 6020                bra.s   left
   22
   23               0000 000C asm_mover         equ *
   24               0000 000C asm_moveright     equ *
   25      0000000C 205F                movea.l (sp)+,return
   26      0000000E 201F                move.l  (sp)+,length
   27      00000010 6F0C                ble.s   nonpos                  length must be > 0
   28      00000012 245F                movea.l (sp)+,destination
   29      00000014 225F                movea.l (sp)+,source
   30      00000016 606A                bra.s   right
   31
   32               0000 0018 asm_fastmove      equ *
   33      00000018 205F                movea.l (sp)+,return
   34      0000001A 201F                move.l  (sp)+,length
   35      0000001C 6E04                bgt.s   pos                     length must be > 0
   36      0000001E 504F        nonpos  addq    #8,sp
   37      00000020 4ED0                jmp     (return)
   38      00000022 245F        pos     movea.l (sp)+,destination
   39      00000024 225F                movea.l (sp)+,source
   40
   41      00000026 B5C9                cmpa.l  source,destination      test direction
   42      00000028 6258                bhi.s   right
   43      0000002A 6744                beq.s   done
   44
   45      0000002C 3209        left    move    source,d1               copies for testing oddness
   46      0000002E 340A                move    destination,d2
   47
   48      00000030 E249                lsr     #1,d1           test source address
   49      00000032 640C                bcc.s   l1
   50      00000034 E24A                lsr     #1,d2           source odd, test destination
   51      00000036 643C                bcc.s   slowleft        source odd, destination even, worst case
   52      00000038 14D9                move.b  (source)+,(destination)+        both odd, move 1 initial byte
   53      0000003A 5380                subq.l  #1,length
   54      0000003C 6E06                bgt.s   fastleft
   55      0000003E 4ED0                jmp     (return)
   56
   57      00000040 E24A        l1      lsr     #1,d2           source even, test destination
   58      00000042 6530                bcs.s   slowleft        source even, destination odd, worst case
```



```
   59
   60                           * both source and destination addresses even, word moves possible
   61
   62      00000044 7220        fastleft moveq  #32,delta
   63      00000046 9081                sub.l   delta,length
   64      00000048 6D0E                blt.s   l11
   65      0000004A 4CD9 18FC   lloop1  movem.l (source)+,d2-d7/a3-a4            move 32 byte chunks
   66      0000004E 48D2 18FC           movem.l d2-d7/a3-a4,(destination)
   67      00000052 D4C1                adda    delta,destination
   68      00000054 9081                sub.l   delta,length
   69      00000056 6CF2                bge.s   lloop1
   70
   71      00000058 D07C 001C   l11     add     #28,length
   72      0000005C 6D06                blt.s   l12
   73      0000005E 24D9        lloop2  move.l  (source)+,(destination)+        move 4 byte chunks
   74      00000060 5940                subq    #4,length
   75      00000062 6CFA                bge.s   lloop2
   76
   77      00000064 5440        l12     addq    #2,length
   78      00000066 6D02                blt.s   l13
   79      00000068 34D9                move.w  (source)+,(destination)+        move 2 bytes
   80
   81      0000006A E248        l13     lsr     #1,length
   82      0000006C 6402                bcc.s   done
   83      0000006E 14D9                move.b  (source)+,(destination)+        move odd byte
   84
   85      00000070 4ED0        done    jmp     (return)
   86
   87      00000072 14D9        lloop3  move.b  (source)+,(destination)+        bytes not on word boundaries
   88      00000074 51C8 FFFC   slowleft dbra   length,lloop3
   89      00000078 90BC 0001           sub.l   #65536,length
               0000
   90      0000007E 64F2                bcc.s   lloop3
   91      00000080 4ED0                jmp     (return)
   92
   93
   94      00000082 D3C0        right   adda.l  length,source
   95      00000084 D5C0                adda.l  length,destination
   96
   97      00000086 3209                move    source,d1               copies for testing oddness
   98      00000088 340A                move    destination,d2
   99
  100      0000008A E249                lsr     #1,d1           test source address
  101      0000008C 640C                bcc.s   r1
  102      0000008E E24A                lsr     #1,d2           source odd, test destination
  103      00000090 643C                bcc.s   slowright       source odd, destination even, worst case
  104      00000092 1521                move.b  -(source),-(destination)        both odd, move 1 initial byte
  105      00000094 5380                subq.l  #1,length
  106      00000096 6E06                bgt.s   fastright
  107      00000098 4ED0                jmp     (return)
  108
  109      0000009A E24A        r1      lsr     #1,d2           source even, test destination
  110      0000009C 6530                bcs.s   slowright       source even, destination odd, worst case
  111
  112                           * both source and destination addresses even, word moves possible
  113
  114      0000009E 7220        fastright moveq #32,delta
```

```
PAGE   3 [2.0] 11/4/82 13:14:10  ASSEMBLY OF FASTMOVE.TEXT

  115     000000A0 9081               sub.l   delta,length
  116     000000A2 6D0E               blt.s   rr1
  117     000000A4 92C1       rloop1  suba    delta,source
  118     000000A6 4CD1 18FC          movem.l (source),d2-d7/a3-a4            move 32 byte chunks
  119     000000AA 48E2 3F18          movem.l d2-d7/a3-a4,-(destination)
  120     000000AE 9081               sub.l   delta,length
  121     000000B0 6CF2               bge.s   rloop1
  122
  123     000000B2 D07C 001C rr1      add     #28,length
  124     000000B6 6D06               blt.s   rr2
  125     000000B8 2521       rloop2  move.l  -(source),-(destination)        move 4 byte chunks
  126     000000BA 5940               subq    #4,length
  127     000000BC 6CFA               bge.s   rloop2
  128
  129     000000BE 5440       rr2     addq    #2,length
  130     000000C0 6D02               blt.s   rr3
  131     000000C2 3521               move.w  -(source),-(destination)        move 2 bytes
  132
  133     000000C4 E248       rr3     lsr     #1,length
  134     000000C6 64A8               bcc.s   done
  135     000000C8 1521               move.b  -(source),-(destination)        move odd byte
  136     000000CA 4ED0               jmp     (return)
  137
  138     000000CC 1521       rloop3  move.b  -(source),-(destination)        bytes not on word boundaries
  139     000000CE 51C8 FFFC slowright dbra   length,rloop3
  140     000000D2 90BC 0001          sub.l   #65536,length
                   0000
  141     000000D8 64F2               bcc.s   rloop3
  142     000000DA 4ED0               jmp     (return)
  143
  144                                 nosyms
  145                                 end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

FMINIT

Purpose

FMINIT is the initialization driver for the 9885 disc.

Usage

FMINIT is used by MEDIAINIT for 9885 discs.

```
PAGE   1 [2.0] 11/4/82 14:30:20  ASSEMBLY OF FMINIT.TEXT
PASS 1 COMPLETE. ERRORS: 0
    2
    3
    4    00000000                    rorg    0
    5                                nosyms
    6
    7
    8                           ************************************************
    9                           *            PASCAL module interface           *
   10                           ************************************************
   11
   12                            src    module Fminit;
   13                            src
   14                            src    import
   15                            src      sysglobals,
   16                            src      midecs;
   17                            src
   18                            src    export
   19                            src      function Fintdata: interleave_data;
   20                            src      function Fphydata: physical_data;
   21                            src      procedure Fminitialize(port: charptr; un: byte; intlve: shortint);
   22                            src
   23                            src    end;
   24
   25                           *
   26                           * def's
   27                           *
   28                                   def     Fminit_Fminit       module intialization routine
   29                                   def     Fminit_Fintdata     interleave factor data function
   30                                   def     Fminit_Fphydata     physical attributes data function
   31                                   def     Fminit_Fminitialize 9885 formatting routine
   32
   33                           *
   34                           * ref's
   35                           *
   36                                   refa    sysglobals
   37
   38                           *
   39                           * module intialization routine
   40                           *
   41    00000000 4E75          Fminit_Fminit   rts
   42
```

```
PAGE   2 [2.0] 11/4/82 14:30:20 FMINIT - 9885 INITIALIZATION ROUTINE
   44
   45                           ************************************************
   46                           *                  functions                   *
   47                           ************************************************
   48
   49                           *
   50                           * interleave factor data function
   51                           *
   52    00000002 0001          intdata dc      1       minimum interleave factor
   53    00000004 001D                  dc      29      maximum interleave factor
   54    00000006 0001                  dc      1       default interleave factor
   55
   56              0000 0008 Fminit_Fintdata equ *
   57    00000008 205F                  movea.l (sp)+,a0                return address
   58    0000000A 225F                  movea.l (sp)+,a1                return variable address
   59    0000000C 4CBA 0007             movem   intdata,d0-d2           interleave_data record constant
                  FFF2
   60    00000012 4891 0007             movem   d0-d2,(a1)              assign the return variable
   61    00000016 4ED0                  jmp     (a0)                    return
   62
   63
   64
   65                           *
   66                           * physical attributes data function
   67                           *
   68    00000018 0000 003F phydata dc.l    67-4    number of tracks per surface
   69    0000001C 0000 0001         dc.l    1       number of surfaces per media
   70    00000020 0000 001E         dc.l    30      number of sectors per track
   71
   72              0000 0024 Fminit_Fphydata equ *
   73    00000024 205F                  movea.l (sp)+,a0                return address
   74    00000026 225F                  movea.l (sp)+,a1                return variable address
   75    00000028 4CFA 0007             movem.l phydata,d0-d2           interleave_data record constant
                  FFEC
   76    0000002E 48D1 0007             movem.l d0-d2,(a1)              assign the return variable
   77    00000032 4ED0                  jmp     (a0)                    return
   78
```

```
                        ************************************************
                        *      passed parameters & local variables     *
                        ************************************************

                        *
                        * local variables
                        *
            FFFF FEAA locals  equ     -342                    total local area size

            FFFF FEAA v85buf  equ     -342        (256 bytes) read/write buffer

            FFFF FFAA v85ibuf equ     -86         (62 bytes)  interleave buffer

            FFFF FFE8 v85lrp  equ     -24         (long)      logical record pointer
            FFFF FFEC v85lrso equ     -20         (long)      logical record spiral offset

            FFFF FFF0 v85r29o equ     -16         (word)      logical record 29 offset
            FFFF FFF2 v85frec equ     -14         (word)      first record
            FFFF FFF4 v85lr0  equ     -12         (word)      this track's logical record 0
            FFFF FFF6 v85scmd equ     -10         (word)      skeleton command
            FFFF FFF8 v85stat equ     -8          (word)      status word

            FFFF FFFA v85rc   equ     -6          (byte)      record count
            FFFF FFFB v85rwf  equ     -5          (byte)      read/write flag
            FFFF FFFC v85gtc  equ     -4          (byte)      good track count
            FFFF FFFD v85btc  equ     -3          (byte)      bad track count
            FFFF FFFE v85patc equ     -2          (byte)      test pattern count
            FFFF FFFF v85gdcb equ     -1          (byte)      gpio dma control byte

                        *
                        * passed parameters
                        *
            0000 0000 oldA6   equ     0           (long)      dynamic link
            0000 0004 retaddr equ     4           (long)      return address
            0000 0008 intlve  equ     8           (word)      interleave factor
            0000 000A un      equ     10          (word)      unit number
            0000 000C port    equ     12          (long)      card port address
```



```
                        ************************************************
                        *                escape sequences              *
                        ************************************************

                        *
                        * assignment of ioresult values
                        *
            0000 0010 ior_notGPIO     equ     16      (znodevice)             card is not GPIO
            0000 0010 ior_not9885     equ     16      (znodevice)             peripheral is not 9885
            0000 002E ior_mediachange equ     46      (zmediumchanged)        media has been changed
            0000 0011 ior_initfailed  equ     17      (zinitfail)             initialization failed
            0000 0015 ior_catchall    equ     21      (zcatchall)             error undetermined

                        *
                        * 9885 error code mapping to ioresult values
                        *
00000034 15         f85ecm  dc.b    21  (zcatchall)         (0)     no error
00000035 10                 dc.b    16  (znodevice)         (1)     not all drives powered
00000036 22                 dc.b    34  (znomedium)         (2)     door open
00000037 22                 dc.b    34  (znomedium)         (3)     no disc in drive
00000038 12                 dc.b    18  (zprotected)        (4)     badcommand (write protected)
00000039 20                 dc.b    32  (znoblock)          (5)     record header error
0000003A 20                 dc.b    32  (znoblock)          (6)     track not found
0000003B 01                 dc.b     1  (zbadblock)         (7)     data checkword error
0000003C 14                 dc.b    20  (zbadhardware)      (8)     data overrun
0000003D 01                 dc.b     1  (zbadblock)         (9)     verify failed

                        *
                        * subroutine ioresult escape: enter with ioresult in d0.1
                        *
0000003E 2B40 FFEA ioresc   move.l  d0,sysglobals-22(a5)    store the ioresult
00000042 3B7C FFF6          move    #-10,sysglobals-2(a5)   store the escapecode
         FFFE
00000048 4E4A               trap    #10                     escape a'la' Pascal

                        *
                        * subroutine decode and issue the error code
                        *
0000004A 7200       f85die  moveq   #0,d1                   clear upper byte for word indexing
0000004C 122E FFF8          move.b  v85stat(a6),d1          9885 error code

00000050 7015               moveq   #ior_catchall,d0        ioresult in case error code is out of range
00000052 B27C 0009          cmp     #9,d1                   is the error code within the expected range?
00000056 62E6               bhi     ioresc                  branch if not

00000058 103B 10DA          move.b  f85ecm(d1),d0           load the appropriate ioresult value
0000005C 60E0               bra     ioresc                  escape with the bad ioresult
```

```
169
170                     ************************************************************************
171                     *                  gpio interface card switch settings                 *
172                     ************************************************************************
173
174                     *            option switches
175
176                     *    1) invert pclt             open
177                     *    2) invert pflg             open
178                     *    3) invert psts             open
179                     *    4) full/pulse handshake    open
180                     *    5) invert data in          open
181                     *    6) invert data out         open
182
183
184                     *          data clock switches
185
186                     *    1) read                    open            \
187                     *    2) ready to busy                 closed      lower data register
188                     *    3) busy to ready           open            /
189                     *    4) read                    open            \
190                     *    5) ready to busy                 closed      upper data register
191                     *    6) busy to ready           open            /
192
193
194                     *    select code switches: 0-31; default 1
195
196
197                     *    interrupt priority switches: don't cares; default 0
198
```



```
200
201                     ************************************************************************
202                     *                                                                      *
203                     * basic program flow is as follows:                                    *
204                     *                                                                      *
205                     *  Generate the interleave buffer containing the logical record        *
206                     *  number sequences.                                                   *
207                     *                                                                      *
208                     *  Reset dumbo to force a "seek" to physical track 0.                  *
209                     *                                                                      *
210                     *  Format & test the current track. If the track is good, increment    *
211                     *  the good track count. If not, increment the bad track count and     *
212                     *  make the bad track invisible. Step the heads in one track. Repeat   *
213                     *  until all 67 tracks have been processed.                            *
214                     *                                                                      *
215                     *  Reset dumbo again.                                                  *
216                     *                                                                      *
217                     *  If more than 4 tracks were bad, issue an error.                     *
218                     *                                                                      *
219                     ************************************************************************
220
221
222
223                     ************************************************************************
224                     *                                                                      *
225                     * procedure for formatting and testing one track                       *
226                     *                                                                      *
227                     *   1. Format the track with the sequence of logical record numbers    *
228                     *      appropriate for the specified interleave factor.                *
229                     *                                                                      *
230                     *   2. With tight margins read back the data written by the format.    *
231                     *      Write & read back under tight margins 3 other test patterns.    *
232                     *      Write 0's to all records.                                       *
233                     *                                                                      *
234                     *   3. If the track had no errors, increment the good track count.     *
235                     *      Else increment the bad track count and make the (bad) track     *
236                     *      invisible.                                                      *
237                     *                                                                      *
238                     ************************************************************************
239
```

```
                        ********************************************************************
                        *                          format the media                        *
                        ********************************************************************

                        *
                        * preliminary setup & checks
                        *
              0000 005E Fminit_Fminitialize equ     *

0000005E 4E56 FEAA              link    a6,#locals              build our stack frame

00000062 286E 000C              movea.l port(a6),a4             a4 dedicated as the gpio card pointer
                        *
                        * test for gpio card; status clear; sti0 & sti1 clear
                        *
00000066 7010                   moveq   #ior_notGPIO,d0         ioresult in case card is not GPIO
00000068 721F                   moveq   #31,d1                  mask for card id bits
0000006A C22C 0001              and.b   1(a4),d1                isolate the card id bits
0000006E 5701                   subq.b  #3,d1                   is the card GPIO?
00000070 66CC                   bne     ioresc                  escape if not

00000072 7010                   moveq   #ior_not9885,d0         ioresult in case peripheral is not a 9885
00000074 720B                   moveq   #$0B,d1                 mask for peripheral status bits
00000076 C22C 0007              and.b   7(a4),d1                isolate the appropriate status bits
0000007A 66C2                   bne     ioresc                  branch if inappropriate status for 9885
                        *
                        * test for media change
                        *
0000007C 197C 0000              move.b  #0,3(a4)                clear enab, word, dmac1, dmac0
              0003
00000082 197C 0000              move.b  #0,7(a4)                clear ctl0 & ctl1
              0007

00000088 6100 0392              bsr     f85lgin                 give the password

0000008C 70FC                   moveq   #-4,d0                  build the
0000008E 806E 000A              or      un(a6),d0                 request status
00000092 E858                   ror     #4,d0                       command
00000094 6100 038A              bsr     f85wo                   issue it

00000098 6100 0390              bsr     f85wi                   fetch the status word
0000009C 3D40 FFF8              move    d0,v85stat(a6)          save the status word for now

000000A0 7000                   moveq   #0,d0
000000A2 6100 037C              bsr     f85wo                   complete the handshake with the '85

000000A6 702E                   moveq   #ior_mediachange,d0     ioresult in case media was changed

000000A8 7204                   moveq   #$04,d1                 disc changed bit
000000AA C26E FFF8              and     v85stat(a6),d1          is it set?
000000AE 668E                   bne     ioresc                  escape if so
```



```
                        *
                        * generate the interleave buffer (in ibuf)
                        *
000000B0 41EE FFAA              lea     v85ibuf(a6),a0          buffer's first byte address
000000B4 43E8 001E              lea     30(a0),a1               buffer's last byte address plus one
000000B8 70FF                   moveq   #-1,d0                  buffer's initial contents
000000BA 7207                   moveq   #8-1,d1                 loop counter
000000BC 20C0                   move.l  d0,(a0)+                store four bytes
000000BE 51C9 FFFC              dbra    d1,*-2                  loop until done

000000C2 41EE FFAA              lea     v85ibuf(a6),a0          buffer's first byte address
000000C6 4210                   clr.b   (a0)                    logical record 0 goes here
000000C8 7001                   moveq   #1,d0                   logical record number counter

000000CA D0EE 0008 f85gbl       adda    intlve(a6),a0           bump by the interleave factor

000000CE B1C9      f85tpr       cmpa.l  a1,a0                   are we still in range?
000000D0 6D04                   blt.s   *+6                     branch if so
000000D2 90FC 001E              suba    #30,a0                  otherwise, circle around

000000D6 4A18                   tst.b   (a0)+                   is this slot empty?
000000D8 6AF4                   bpl     f85tpr                  branch if not

000000DA 1100                   move.b  d0,-(a0)                this slot's empty: claim it
000000DC 5240                   addq    #1,d0                   bump the logical record number
000000DE B07C 001E              cmp     #30,d0                  all 30 lrn's (0-29) placed?
000000E2 6DE6                   blt     f85gbl                  loop until done

000000E4 43EE FFAA              lea     v85ibuf(a6),a1          buffer's FBA (lrn 0's position)
000000E8 91C9                   suba.l  a1,a0                   offset from lrn 0 to lrn 29
000000EA 3D48 FFF0              move    a0,v85r29o(a6)          save it
                        *
                        * make another copy of the buffer
                        *
000000EE 4CEE 00FF              movem.l v85ibuf(a6),d0-d7       load 32 bytes
              FFAA
000000F4 48EE 00FF              movem.l d0-d7,v85ibuf+30(a6)    store 32 bytes
              FFC8
                        *
                        * some initializations for the main loop
                        *
000000FA 1940 0001              move.b  d0,1(a4)                clear the gpio card & reset dumbo

000000FE 41EE FFAA              lea     v85ibuf(a6),a0          interleave buffer's FBA
00000102 2D48 FFEC              move.l  a0,v85lrso(a6)          init the logical record spiral offset
00000106 422E FFFC              clr.b   v85gtc(a6)              clear the good track count
0000010A 422E FFFD              clr.b   v85btc(a6)              clear the bad track count

0000010E 700C                   moveq   #$0C,d0
00000110 806E 000A              or      un(a6),d0               unit number
00000114 E858                   ror     #4,d0                   skeleton for various commands
00000116 3D40 FFF6              move    d0,v85scmd(a6)          save it
```

```
                        *
                        * main loop section
                        *
0000011A 6150           f85ml   bsr.s   f85fmt                  format and test one track

0000011C 102E FFFC              move.b  v85gtc(a6),d0           good track count
00000120 D02E FFFD              add.b   v85btc(a6),d0           bad track count
00000124 B03C 004D              cmp.b   #77,d0                  all done?
00000128 6C22                   bge.s   f85fc                   branch if so

0000012A 6100 02F0              bsr     f85lgin                 give the password
0000012E 302E FFF6              move    v85scmd(a6),d0          basic command skeleton
00000132 807C 0F9F              or      #$0F9F,d0               form a step in command
00000136 6100 02E8              bsr     f85wo                   issue it

0000013A 6100 018E              bsr     f85fst                  fetch status
0000013E C07C FFFC              and     #$FFFC,d0               strip off the drive bits
00000142 B07C 0020              cmp     #$0020,d0               seek complete, no errors?
00000146 67D2                   beq     f85ml                   if so, continue with the main loop
00000148 6000 FF00              bra     f85die                  otherwise, decode and issue error


                        *
                        * formatting complete: wrap it up
                        *
0000014C 1940 0001 f85fc        move.b  d0,1(a4)                clear the gpio card & reset dumbo

00000150 7011                   moveq   #ior_initfailed,d0      ioresult in case of excessive rejected tracks
00000152 0C2E 003F              cmpi.b  #63,v85gtc(a6)          did we get enough good tracks?
         FFFC
00000158 6D00 FEE4              blt     ioresc                  escape if not

0000015C 4E5E                   unlk    a6                      remove our stack frame
0000015E 205F                   movea.l (sp)+,a0                pop off the return address
00000160 508F                   addq.l  #8,sp                   pop off the parameters
00000162 4ED0                   jmp     (a0)                    return


                        *
                        * test patterns (the first is written by the format command!)
                        *                       1                   2
                        *       dc      $C6C6   1100011011000110110001101100011
00000164 6363           f85ptrn dc      $6363   0110001101100011011000110110001
00000166 DB6D                   dc      $DB6D   1101101101101101101101101101101
00000168 8888                   dc      $8888   1000100010001000100010001000100
0000016A 0000                   dc      $0000   (final sector contents)
```



```
                        *
                        * routine to format and test one track
                        *
0000016C 102E FFFC f85fmt       move.b  v85gtc(a6),d0           good track count
00000170 B03C 003F              cmp.b   #63,d0                  already have enough good tracks?
00000174 6C00 011E              bge     f85mtd                  branch if so (don't leave "extra" tracks)

00000178 D02E FFFD              add.b   v85btc(a6),d0           compute current physical track number
0000017C B03C 0042              cmp.b   #66,d0                  out past the 9885's "supported" range?
00000180 6E00 0112              bgt     f85mtd                  branch if so (don't leave "extra" tracks)

00000184 6100 0296              bsr     f85lgin                 give the password
00000188 302E FFF6              move    v85scmd(a6),d0          basic command skeleton
0000018C EA58                   ror     #5,d0
0000018E 802E FFFC              or.b    v85gtc(a6),d0           current logical track number
00000192 EB58                   rol     #5,d0
00000194 807C 001E              or      #30,d0                  format command for this track

00000198 6100 027A              bsr     f85wf                   wait for the flag
0000019C 3940 0004              move    d0,4(a4)                place the command in the output buffer
000001A0 4E4B                   trap    #11                                                  scs
                        * scs   move    sr,-(sp)                prepare to disable interrupts
000001A2 007C 2700              ori     #$2700,sr               disable interrupts
                        ********************** interrupts disabled **************************
000001A6 1880                   move.b  d0,(a4)                 set the peripheral control flag

000001A8 721D                   moveq   #30-1,d1                initialize the loop counter
000001AA 206E FFEC              movea.l v85lrso(a6),a0          logical record spiral offset
000001AE 7000                   moveq   #0,d0                   need upper byte cleared

000001B0 1018           f85flb  move.b  (a0)+,d0                next logical record number to send

000001B2 6100 024C              bsr     f85cswf                 check status and wait for the flag
000001B6 600A                     bra.s f85fle                  ret 1; terminate & check status

000001B8 3940 0004              move    d0,4(a4)                place the lrn in the output buffer
000001BC 1880                   move.b  d0,(a4)                 set the peripheral control flag

000001BE 51C9 FFF0              dbra    d1,f85flb               loop until all 30 lrn's sent

000001C2 46DF           f85fle  move    (sp)+,sr                re-enable interrupts
                        ********************** interrupts re-enabled ************************
000001C4 6100 0104              bsr     f85fst                  fetch status
000001C8 C07C FFF8              and     #$FFF8,d0               strip the drive & disc change bits
000001CC B07C 0040              cmp     #$0040,d0               format complete, no errors?
000001D0 6600 FE78              bne     f85die                  if not, decode and issue error
                        *
                        * prepare for reading and writing under i/o control
                        *
000001D4 47FA FF8E              lea     f85ptrn,a3              first test pattern address

000001D8 1D7C 0004              move.b  #4,v85patc(a6)          init the pattern count
         FFFE
```

```
                        *
                        * loop to read verify the previous pattern and write the next pattern
                        *   note: the first test pattern was written by the format command!
                        *         the last pattern written is 0, and will not be read verified
                        *
                        *
                        * verify under i/o control
                        *
000001DE 206E FFEC f85vwl  movea.l v85lrso(a6),a0          logical record spiral offset
000001E2 2D48 FFE8         move.l  a0,v85lrp(a6)           init the logical record pointer

000001E6 1D7C 001E         move.b  #30,v85rc(a6)           init the record counter
         FFFA

000001EC 7000              moveq   #0,d0                   clear the upper byte
000001EE 102E FFFC         move.b  v85gtc(a6),d0           this track's logical track#
000001F2 C0FC 001E         mulu    #30,d0                  this track's logical record 0
000001F6 3D40 FFF4         move    d0,v85lr0(a6)           save it

000001FA 206E FFE8 f85vl   movea.l v85lrp(a6),a0           logical record pointer
000001FE 7000              moveq   #0,d0                   clear upper byte and word
00000200 1018              move.b  (a0)+,d0                logical record to verify next
00000202 D06E FFF4         add     v85lr0(a6),d0           this track's logical record 0
00000206 2D48 FFE8         move.l  a0,v85lrp(a6)           updated logical record pointer
0000020A 3D40 FFF2         move    d0,v85frec(a6)          record to verify

0000020E 51EE FFFB         sf      v85rwf(a6)              set read/write flag to read
00000212 6100 00D0         bsr     f85xfr                  do the transfer
00000216 6042              bra.s f85ferr                   ret 1; some error occurred

00000218 532E FFFA         subq.b  #1,v85rc(a6)            decrement the record counter
0000021C 6EDC              bgt     f85vl                   loop til all 30 records are verified

                        *
                        * write under i/o control
                        *
0000021E 3013              move    (a3),d0                 test pattern
00000220 4840              swap    d0
00000222 301B              move    (a3)+,d0                need it in upper & lower words
00000224 41EE FEAA         lea     v85buf(a6),a0           first word address
00000228 723F              moveq   #64-1,d1                initialize the loop counter
0000022A 20C0              move.l  d0,(a0)+                write four bytes
0000022C 51C9 FFFC         dbra    d1,*-2                  loop until done

00000230 206E FFEC         movea.l v85lrso(a6),a0          logical record spiral offset
00000234 2D48 FFE8         move.l  a0,v85lrp(a6)           logical record pointer

00000238 1D7C 001E         move.b  #30,v85rc(a6)           init the record count
         FFFA
```



```
0000023E 206E FFE8 f85wl   movea.l v85lrp(a6),a0           logical record pointer
00000242 7000              moveq   #0,d0                   clear upper byte and word
00000244 1018              move.b  (a0)+,d0                next record to be written
00000246 D06E FFF4         add     v85lr0(a6),d0           this track's logical record 0
0000024A 2D48 FFE8         move.l  a0,v85lrp(a6)           updated logical record pointer
0000024E 3D40 FFF2         move    d0,v85frec(a6)          record number to write

00000252 50EE FFFB         st      v85rwf(a6)              set the read/write flag to write
00000256 6100 008C         bsr     f85xfr                  do the transfer
0000025A 6020              bra.s f85err                    ret 1; some error occurred

0000025C 532E FFFA         subq.b  #1,v85rc(a6)            decrement the record count
00000260 6EDC              bgt     f85wl                   loop till all 30 records are written


00000262 532E FFFE         subq.b  #1,v85patc(a6)          decrement the pattern count
00000266 6ED0 FF76         bgt     f85vwl                  loop until done

0000026A 522E FFFC         addq.b  #1,v85gtc(a6)           increment the good track count

0000026E 701A              moveq   #26,d0
00000270 906E 0008         sub     intlve(a6),d0
00000274 908E FFF0         sub     v85r29o(a6),d0          (offset=offset-inter-r29o+26)
00000278 48C0              ext.l   d0                      need the offset long
0000027A 6038              bra.s   f85bof                  bump the logical record offset


                        *
                        * an error occurred in verifying or writing: is it "ok"?
                        *
0000027C 102E FFF8 f85err  move.b  v85stat(a6),d0          the upper byte of the error code word

00000280 5B00              subq.b  #5,d0                   id error?
00000282 6710              beq.s   f85mtd                  branch if so
00000284 5300              subq.b  #6-5,d0                 track error?
00000286 670C              beq.s   f85mtd                  branch if so
00000288 5300              subq.b  #7-6,d0                 crc error?
0000028A 6708              beq.s   f85mtd                  branch if so
0000028C 5500              subq.b  #9-7,d0                 verify error?
0000028E 6704              beq.s   f85mtd                  branch if so

00000290 6000 FDB8         bra     f85die                  take the error exit
```

```
548
549                         *
550                         * mark the (bad) track defective (invisible)
551                         *
552    00000294 6100 0186 f85mtd  bsr     f851gin                 give the password
553
554    00000298 302E FFF6         move    v85scmd(a6),d0          basic command skeleton
555    0000029C 807C 0FBF         or      #$0FBF,d0               form a mark track defective cmd
556    000002A0 6100 017E         bsr     f85wo                   issue it
557
558    000002A4 6124              bsr.s   f85fst                  fetch status
559    000002A6 C07C FFFC         and     #$FFFC,d0               strip off the drive bits
560    000002AA 6600 FD9E         bne     f85die                  branch if any errors
561
562    000002AE 522E FFFD         addq.b  #1,v85btc(a6)           increment the bad track count
563
564    000002B2 70FC              moveq   #-4,d0                  logical record offset to be bumped by 4
565                         *      bra.s   f85bof                  bump the offset
566
567
568                         *
569                         * subroutine to bump the logical record spiral offset
570                         *
571    000002B4 D0AE FFEC f85bof  add.l   v85lrso(a6),d0          bump it
572    000002B8 2D40 FFEC         move.l  d0,v85lrso(a6)          save it
573    000002BC 41EE FFAA         lea     v85ibuf(a6),a0          interleave buffer's FBA
574    000002C0 B088              cmp.l   a0,d0                   offset still in range?
575    000002C2 6C04              bge.s   *+6                     branch if so
576    000002C4 701E              moveq   #30,d0                  else put it in range
577    000002C6 60EC              bra.s   f85bof
578    000002C8 4E75              rts
579
580                         *
581                         * subroutine to fetch status
582                         *
583    000002CA 6100 0134 f85fst  bsr     f85cswf                 check status and wait for the flag
584    000002CE 4E71                nop                           ret 1; error occurred
585
586    000002D0 6100 015A         bsr     f85winw                 word in; don't wait on the flag
587    000002D4 3D40 FFF8         move    d0,v85stat(a6)          save the status word for now
588
589    000002D8 7000              moveq   #0,d0
590    000002DA 6100 0144         bsr     f85wo                   complete the handshake with the '85
591
592    000002DE 302E FFF8         move    v85stat(a6),d0          load the status word
593    000002E2 4E75              rts
594
```



```
596
597                         ************************************************************************
598                         *                        routine to transfer one sector                *
599                         ************************************************************************
600
601                         *
602                         * gain access to the dma channel
603                         *
604    000002E4 247C 0050 f85xfr  movea.l #$500000,a2             DMA channel 0 address
               0000
605    000002EA 7005              moveq   #$05,d0                 gpio dma control byte: chan 0
606                         *
607                         * set up the gpio card
608                         *
609    000002EC 1D40 FFFF         move.b  d0,v85gdcb(a6)          save the gpio dma control byte
610    000002F0 197C 0000         move.b  #0,3(a4)                clear enab, word, dmac1, dmac0
               0003
611    000002F6 197C 0000         move.b  #0,7(a4)                clear ctl1 & ctl0
               0007
612                         *
613                         * build and issue the seek command
614                         *
615    000002FC 6100 011E         bsr     f851gin                 give the password
616
617    00000300 302E FFF2         move    v85frec(a6),d0          first record number
618    00000304 48C0              ext.l   d0                      make it long for division
619    00000306 81FC 001E         divs    #30,d0                  split into track & sector #'s
620    0000030A 4840              swap    d0                      track in upper; sector in lower
621    0000030C EA58              ror     #5,d0                   sector in upper bits of lower
622    0000030E EE98              ror.l   #7,d0                   track&sector in upper bits of lower
623    00000310 806E 000A         or      un(a6),d0               track&sector&00&unit in lower
624    00000314 E458              ror     #2,d0                   unit&track&sector&00
625    00000316 807C 0003         or      #3,d0                   unit&track&sector&11
626    0000031A E458              ror     #2,d0                   11&unit&track&sector (finally!)
627    0000031C 6100 0102         bsr     f85wo                   issue the seek command
628
629                         *
630                         * partially build the appropriate transfer command
631                         *
632    00000320 6100 00FA         bsr     f851gin                 give the password
633
634    00000324 302E 000A         move    un(a6),d0               unit number
635    00000328 E858              ror     #4,d0                   position for read/write/verify
636    0000032A 5240              addq    #1,d0                   specify a record count of one
637
638    0000032C 41EE FEAA         lea     v85buf(a6),a0           read/write buffer's FWA
639    00000330 2488              move.l  a0,(a2)                 write in the DMA channel's address register
640
641    00000332 727F              moveq   #127,d1                 #words-1 in a sector
642    00000334 3541 0004         move    d1,4(a2)                write in the DMA channel's count register
643
644    00000338 6100 00C6         bsr     f85cswf                 check status and wait for the flag
645    0000033C 605A                bra.s f85_grs                 ret 1 - error; go read status
646
647    0000033E 4A2E FFFB         tst.b   v85rwf(a6)              read/write flag
648    00000342 6B56              bmi.s   f85_wrt                 branch if a write
649
```

```
                        *
                        * section for verify (read)
                        *
00000344 08C0 000E              bset    #14,d0                  make a verify command
00000348 3940 0004              move    d0,4(a4)                place cmnd in the output buffer

0000034C 4E4B                   trap    #11                                                     scs
                        * scs   move    sr,-(sp)                prepare to disable interrupts
0000034E 007C 2700              ori     #$2700,sr               disable interrupts (except nmi)
                        ********************** interrupts disabled **************************
00000352 1880                   move.b  d0,(a4)                 set the peripheral control line

00000354 6100 00AA              bsr     f85cswf                 check status and wait for the flag
00000358 600C                     bra.s f85_rby                 ret 1; bypass because of error

0000035A 397C 0000              move    #0,4(a4)                clear the output buffers
              0004
00000360 4A6C 0004              tst     4(a4)                   set direction in w/ dummy read
00000364 1880                   move.b  d0,(a4)                 set the peripheral control flag

00000366 196E FFFF f85_rby      move.b  v85gdcb(a6),3(a4)       set the gpio dma control bits
              0003
0000036C 536A 0004              subq    #1,4(a2)                don't dma the last word in
00000370 6D06                   blt.s   f85_rei                 don't arm if no words to dma in
00000372 357C 0002              move    #$0002,6(a2)            arm the dma channel
              0006
00000378 46DF           f85_rei move    (sp)+,sr                re-enable interrupts
                        ********************** interrupts re-enabled ************************
0000037A 41EE FFA8              lea     v85buf+254(a6),a0       transfer's last word address

0000037E 082C 0003 f85_rwt      btst    #3,7(a4)                peripheral status set?
              0007
00000384 6648                   bne.s   f85_tdx                 if so, terminate dma xfer now!
00000386 082A 0000              btst    #0,7(a2)                dma channel still armed?
              0007
0000038C 66F0                   bne     f85_rwt                 if so, loop

0000038E 6100 0070              bsr     f85cswf                 check status and wait for the flag
00000392 603A                     bra.s f85_tdx                 ret 1 - error; bypass else hang
00000394 30AC 0004              move    4(a4),(a0)              transfer the last word
00000398 6034           f85_grs bra.s   f85_tdx                 go read the status word
```



```
                        *
                        * section for write
                        *
0000039A 08C0 000F f85_wrt      bset    #15,d0                  make it a write command
0000039E 3940 0004              move    d0,4(a4)                place the command in the output buffer

000003A2 4E4B                   trap    #11                                                     scs
                        * scs   move    sr,-(sp)                prepare to disable interrupts
000003A4 007C 2700              ori     #$2700,sr               disable interrupts (except nmi)
                        ********************** interrupts disabled **************************
000003A8 1880                   move.b  d0,(a4)                 set the peripheral control line

000003AA 196E FFFF              move.b  v85gdcb(a6),3(a4)       set gpio card dma control bits
              0003
000003B0 357C 0006              move    #$0006,6(a2)            arm the dma channel
              0006

000003B6 46DF                   move    (sp)+,sr                re-enable interrupts
                        ********************** interrupts re-enabled ************************
000003B8 082C 0003 f85_wwt      btst    #3,7(a4)                peripheral status set?
              0007
000003BE 660E                   bne.s   f85_tdx                 if so, terminate dma xfer now!
000003C0 082A 0000              btst    #0,7(a2)                dma channel still armed?
              0007
000003C6 66F0                   bne.s   f85_wwt                 if so, loop

000003C8 6100 0036              bsr     f85cswf                 check status and wait for the flag
000003CC 4E71                     nop                           ret 1; some error occurred

                        *
                        * terminate the dma transfer and read the status word
                        *
000003CE 3C12           f85_tdx move    (a2),d0                 disarm the dma channel
000003D0 422C 0003              clr.b   3(a4)                   clr the gpio card dma control bits

000003D4 197C 0001              move.b  #1,7(a4)                set the transfer complete bit
              0C07
000003DA 6100 0050              bsr     f85winw                 word in; don't wait for the flag
000003DE 3C40 FFF8              move    d0,v85stat(a6)          save the status word for now
000003E2 197C 0000              move.b  #0,7(a4)                clear the transfer complete bit
              0C07

000003E8 7C00                   moveq   #0,d0
000003EA 6100 0034              bsr     f85wo                   complete the handshake with the '85

                        *
                        * check for errors
                        *
000003EE 3C2E FFF8              move    v85stat(a6),d0          drive ready, seek complete,
000003F2 CC7C FFF0              and     #$FFF0,d0                 transfer complete, &
000003F6 BC7C 0060              cmp     #$0060,d0                   no error in error code?
000003FA 6602                   bne.s   *+4                     branch if any errors
000003FC 5497                   addq.l  #2,(sp)                 setup a normal (ret 2) exit
000003FE 4E75                   rts                             exit
```

```
                        ********************************************************************
                        *                      9885 shared driver routines                 *
                        ********************************************************************
                        *
                        * routine to check status while waiting on the flag
                        *
00000400 5497           f85cswf addq.l  #2,(sp)                 anticipate a normal (ret 2) exit

00000402 0814 0000      f85tfb  btst    #0,(a4)                 flag bit
00000406 660A                   bne.s   f85ret                  branch if set

00000408 082C 0003              btst    #3,7(a4)                peripheral status bit
         0007
0000040E 67F2                   beq.s   f85tfb                  keep looping unless it's set

00000410 5597                   subq.l  #2,(sp)                 change the ret 2 back to a ret 1
00000412 4E75           f85ret  rts                             exit



                        *
                        * routine to wait for the flag
                        *
00000414 0814 0000      f85wf   btst    #0,(a4)                 flag bit
00000418 67FA                   beq     *-4                     loop until set
0000041A 4E75                   rts



                        *
                        * routine to send the 9885 password
                        *
0000041C 303C AE87      f85lgin move    #44679,d0               9885's "secret" password


                        *
                        * routine to send one word out on the gpio card
                        *
00000420 61F2           f85wo   bsr     f85wf                   wait for the flag
00000422 3940 0004              move    d0,4(a4)                place word in the output buffer
00000426 1880                   move.b  d0,(a4)                 set the peripheral control line
00000428 4E75                   rts
```



```
                        *
                        * routine to receive one word in on the gpio card
                        *
0000042A 61E8           f85wi   bsr     f85wf                   wait for the flag

0000042C 397C 0000      f85winw move    #0,4(a4)                clear the output buffer
         0004
00000432 4A6C 0004              tst     4(a4)                   set direction in w/ dummy read
00000436 1880                   move.b  d0,(a4)                 set the peripheral control line
00000438 61DA                   bsr     f85wf                   wait for the flag
0000043A 302C 0004              move    4(a4),d0                load in the word
0000043E 4E75                   rts


                                end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

FORMAT

Purpose

FORMAT is the initialization driver for the 7906, 7920 and 7925 hard discs.

Usage

FORMAT is used solely with MEDIAINIT.

Notes

FORMAT is for HP internal use only; it is not formally supported.

```
PASS 1 COMPLETE. ERRORS: 0
    1                       *
    2                       ************************************************
    3                       *
    4                       * FORMAT Routine for Initialize
    5                       *
    6                       * Last Updated September 30, 1981
    7                       *
    8                       ************************************************
    9                       *
   10    00000000                   RORG    0
   11                               NOSYMS
   12
   13                               DEF     ASMR_FORMAT
   14                               REFA    SYSGLOBALS
   15                               REFA    XMINIT
   16
   17              0000 0010 PHI    EQU     $10
   18              0000 0001 CARD   EQU     $1
   19              0050 0000 DMAC   EQU     $500000  LOCATION OF BUILT IN DMA CARD
   20                       *
   21              FFFF FFEC INITISRIB EQU   XMINIT-20    INIT'S ISRIB
   22                       *
   23                       * PHI REGISTER ADDRESSES, IN ASCENDING ORDER
   24                       *
   25              0000 0011 INTR   EQU     PHI+1
   26              0000 0013 IMSK   EQU     PHI+3
   27              0000 0015 FIFOFFSET EQU  PHI+5
   28              0000 0017 STATPOFFS EQU  PHI+7
   29              0000 0019 CTRL   EQU     PHI+9
   30              0000 001B HPIBADR EQU    PHI+$B
   31              0000 001D PPMSK  EQU     PHI+$D
   32              0000 001F PPSNS  EQU     PHI+$F
   33                       *
   34                       * PHI CARD REGISTERS
   35                       *
   36              0000 0003 CRDCTRL EQU    CARD+2
   37                       *CRDSTAT EQU    CARD+2
   38              0000 0001 ID     EQU     CARD
   39                       *
   40                       * DMA REGISTERS
   41                       *
   42              0050 0000 DMADR0 EQU     DMAC
   43              0050 0004 DMWCT0 EQU     DMAC+4
   44              0050 0006 DMCTRL0 EQU    DMAC+6
   45              0050 0000 DMSTAT EQU     DMAC
   46              0050 0008 DMADR1 EQU     DMAC+8
   47              0050 000C DMWCT1 EQU     DMAC+$C
   48              0050 000E DMCTRL1 EQU    DMAC+$E
   49                       *
   50                       * AMIGO RELATED CONSTANTS
   51                       *
   52              0000 005E PTA    EQU     $5E        PHI TALK ADDRESS
   53              0000 003E PLA    EQU     $3E        PHI LISTEN ADDRESS
   54              0000 0040 DTAC   EQU     $40        DISK PRIMARY TALK BASE
   55              0000 0020 DLAC   EQU     $20        DISK PRIMARY LISTEN BASE
   56              0000 0070 DSJ    EQU     $70        DSJ SECONDARY
   57              0000 0068 DSAD   EQU     $68        DISK STANDARD SECONDARY
   58              0000 0040 ICMND  EQU     $40        INTERFACE COMMAND SETUP
```



```
   59              0000 0080 EOI    EQU     $80        EOI SETUP
   60              0000 005F UNT    EQU     $5F        UNTALK COMMAND
   61              0000 003F UNL    EQU     $3F        UNLISTEN COMMAND
   62                       *
   63                       *
   64              0000 0000 SECTOR EQU     0
   65                       *
   66                       *
   67                       * REGISTER SETUP
   68                       *
   69              0000 0000 DTEMP  EQU     D0
   70              0000 0001 MTEMP  EQU     D1
   71              0000 0002 BCOUNT EQU     D2
   72              0000 0003 TCOUNT EQU     D3
   73              0000 0004 DZERO  EQU     D4
   74              0000 0005 FREE3  EQU     D5              (AVAILABLE FOR USE)
   75              0000 0006 FREE2  EQU     D6              (AVAILABLE FOR USE)
   76              0000 0007 FREE1  EQU     D7              (AVAILABLE FOR USE)
   77
   78              0000 0000 BYTEPTR EQU    A0
   79              0000 0001 FIFO   EQU     A1
   80              0000 0002 PORT   EQU     A2
   81              0000 0003 ATEMP  EQU     A3
   82              0000 0004 STATP  EQU     A4
   83              0000 0005 GLOBAL EQU     A5
   84              0000 0006 BASE   EQU     A6
   85                       *SP     EQU     A7
   86
   87
   88                       *
   89                       *  PARAMETERS PASSED FROM PASCAL
   90                       *
   91
   92              0000 0020 PARAMS EQU     40-8            NUMBER OF BYTES FOR PASSED PARAMETERS
   93              0000 0026 BUSADDR EQU    38
   94              0000 0024 UNIT   EQU     36
   95              0000 0022 HEAD   EQU     34
   96              0000 001E PHIPORT EQU    30
   97              0000 001A TRACKB EQU     26
   98              0000 0016 SECTORS EQU    22
   99              0000 0014 SPD    EQU     20
  100              0000 0010 MADR   EQU     16
  101              0000 000C TRACKA EQU     12              ADDRESS OF TRACK A
  102              0000 0008 ERRTYPE EQU     8
  103                       *RETADDR EQU     4              RETURN ADDRESS
  104                       *STATLNK EQU     0              STATIC LINK
  105                       *
  106                       *  LOCAL PARAMETERS
  107                       *
  108              FFFF FFFF DLA    EQU     -1     DISC LISTEN ADDRESS
  109              FFFF FFFE DTA    EQU     -2     DISC TALK ADDRESS
  110              FFFF FFFD DSJBYTE EQU    -3     DSJ BYTE
  111              FFFF FFFC STAT3  EQU     -4     STORAGE FOR STATUS BYTES
  112              FFFF FFFB STAT2  EQU     -5
  113              FFFF FFFA STAT1  EQU     -6
  114              FFFF FFF9 STAT0  EQU     -7
  115              FFFF FFF8 ADDR3  EQU     -8     STORAGE FOR DISC ADDRESS
```

```
                FFFF FFF7 ADDR2    EQU    -9
                FFFF FFF6 ADDR1    EQU    -10
                FFFF FFF5 ADDR0    EQU    -11
                FFFF FFF4 PMASK    EQU    -12    PPOLL MASK
                FFFF FFF3 IFLAG    EQU    -13    INTERRUPT FLAG
                FFFF FFF2 SPDTEMP  EQU    -14    UPPER 3 BITS=S,P,D
                FFFF FFF0 CYLINDR1 EQU    -16    CYL ADDR FOR SEEK
                FFFF FFEE CYLINDR2 EQU    -18    CYL ADDR FOR ADDR REC
                FFFF FFEA ADDRSTOR EQU    -22    STORAGE FOR SYSTEM INTERRUPT VECTOR
                FFFF FFE6 STATSTOR EQU    -26    STORAGE FOR SYSTEM STATIC LINK
                FFFF FFE6 LOCALS   EQU    -26    TOTAL AMOUNT OF LOCAL STOREAGE
                          *
                          *
00000000 4E56 FFE6 ASMR_FORMAT  LINK    BASE,#LOCALS
00000004 246E 001E         MOVEA.L PHIPORT(BASE),PORT      ADDRESS OF PHI CARD INTERFACE
00000008 43EA 0015         LEA     FIFOFFSET(PORT),FIFO
0000000C 49EA 0017         LEA     STATPOFFS(PORT),STATP
00000010 4284              CLR.L   DZERO
00000012 47FA 03E2         LEA     ISV,ATEMP               SET UP INTERRUPT VECTOR
00000016 2B48 FFF4         MOVE.L  ATEMP,INITISRIB+8(GLOBAL)  ADDRESS OF ISR
0000001A 2B4E FFF8         MOVE.L  BASE,INITISRIB+12(GLOBAL)  STATIC LINK
0000001E 122E 0027         MOVE.B  BUSADDR+1(BASE),MTEMP   CALCULATE PPOLL MASK
00000022 103C 0080         MOVE.B  #$80,DTEMP
00000026 E228              LSR.B   MTEMP,DTEMP
00000028 1D40 FFF4         MOVE.B  DTEMP,PMASK(BASE)
0000002C 103C 0040         MOVE.B #DTAC,DTEMP              CALCULATE DISC TALK ADDRESS
00000030 D02E 0027         ADD.B  BUSADDR+1(BASE),DTEMP
00000034 1D40 FFFE         MOVE.B DTEMP,DTA(BASE)
00000038 103C 0020         MOVE.B #DLAC,DTEMP              CALCULATE DISC LISTEN ADDRESS
0000003C D02E 0027         ADD.B  BUSADDR+1(BASE),DTEMP
00000040 1D40 FFFF         MOVE.B DTEMP,DLA(BASE)
00000044 266E 0008         MOVEA.L ERRTYPE(BASE),ATEMP     LOAD ERRTYPE ADDRESS
00000048 2684              MOVE.L  DZERO,(ATEMP)           CLEAR ERRTYPE
0000004A 6100 0378         BSR  SFM     SET FILE MASK
0000004E 266E 000C         MOVEA.L TRACKA(BASE),ATEMP      COMPARE TRACK A & B
00000052 2013              MOVE.L (ATEMP),DTEMP
00000054 B0AE 001A         CMP.L  TRACKB(BASE),DTEMP
00000058 6600 002C         BNE    SPARE
                     *    IF EQUAL THEN FORMAT TRACK A
0000005C 6100 009A         BSR    DMSETUP
00000060 266E 000C         MOVEA.L TRACKA(BASE),ATEMP
00000064 3D6B 0002         MOVE.W 2(ATEMP),CYLINDR1(BASE)
                FFF0
0000006A 302E 0014         MOVE.W SPD(BASE),DTEMP         SHIFT S,P,D TO UPPER 3 BITS
0000006E C1FC 0020         MULS   #32,DTEMP
00000072 1D40 FFF2         MOVE.B DTEMP,SPDTEMP(BASE)
00000076 6100 0238         BSR    SEEK
0000007A 6100 0272         BSR    INITZE
0000007E 6100 0092         BSR    ERCHK
```



```
00000082 6000 006A         BRA    END_IO

                     * NOW SPARE TRACK A WITH TRACK B

00000086 266E 000C SPARE   MOVEA.L TRACKA(BASE),ATEMP
0000008A 3D6B 0002         MOVE.W 2(ATEMP),CYLINDR1(BASE)
                FFF0
00000090 3D6E 001C         MOVE.W TRACKB+2(BASE),CYLINDR2(BASE)
                FFEE
00000096 1D7C 0020         MOVE.B  #32,SPDTEMP(BASE)   SET D-BIT
                FFF2
0000009C 6100 0212         BSR    SEEK
000000A0 6100 02CA         BSR    ADDREC
000000A4 6100 0052         BSR    DMSETUP
000000A8 6100 0244         BSR    INITZE
000000AC 6100 0064         BSR    ERCHK
000000B0 102E FFFD         MOVE.B DSJBYTE(BASE),DTEMP
000000B4 6600 0038         BNE    END_IO
000000B8 102E FFF9         MOVE.B STAT0(BASE),DTEMP
000000BC 0200 001F         ANDI.B #$1F,DTEMP   CLEAR SPD
000000C0 6600 002C         BNE    END_IO
000000C4 3D6E 001C         MOVE.W TRACKB+2(BASE),CYLINDR1(BASE)
                FFF0
000000CA 266E 000C         MOVEA.L TRACKA(BASE),ATEMP
000000CE 3D6B 0002         MOVE.W 2(ATEMP),CYLINDR2(BASE)
                FFEE
000000D4 1D7C 0080         MOVE.B  #128,SPDTEMP(BASE)  SET S-BIT
                FFF2
000000DA 6100 01D4         BSR    SEEK
000000DE 6100 028C         BSR    ADDREC
000000E2 6100 0014         BSR    DMSETUP
000000E6 6100 0206         BSR    INITZE
000000EA 6100 0026         BSR    ERCHK
000000EE 4E5E      END_IO  UNLK   BASE
000000F0 265F              MOVEA.L (SP)+,ATEMP
000000F2 DEFC 0020         ADDA.W  #PARAMS,SP
000000F6 4ED3              JMP     (ATEMP)
                     *
                     *
                     * DMA SETUP ROUTINE
                     * - AT THE END OF THIS ROUTINE THE DMA CARD WILL BE
                     *   SET UP FOR A DISK TRANSFER
                     *
000000F8 23EE 0010 DMSETUP   MOVE.L   MADR(BASE),DMADR0
                0050 0000
00000100 202E 0016           MOVE.L   SECTORS(BASE),DTEMP
00000104 C1FC 0100           MULS     #$100,DTEMP    CONVERT TO BYTE COUNT
00000108 5340                SUBQ     #1,DTEMP
0000010A 33C0 0050           MOVE.W   DTEMP,DMWCT0  LOWER 2 BYTES ONLY
                0004
00000110 4E75                RTS
                     *
                     *
                     * Error Check Routine
                     *
```

```
 221    00000112 6100 0042 ERCHK     BSR      DSJR
 222    00000116 6100 007E           BSR      DSTAT
 223    0000011A 102E FFFD           MOVE.B   DSJBYTE(BASE),DTEMP
 224    0000011E 6600 000E           BNE      ERROR    CHECK DSJ BYTE
 225    00000122 102E FFF9           MOVE.B   STAT0(BASE),DTEMP
 226    00000126 0200 001F           ANDI.B   #$1F,DTEMP   CLEAR SPD BITS
 227    0000012A 6700 0028           BEQ      FINISH   CHECK STATUS 1
 228    0000012E 6100 00D4 ERROR     BSR      RADDREC
 229    00000132 266E 0008           MOVEA.L  ERRTYPE(BASE),ATEMP
 230    00000136 16EE FFF9           MOVE.B   STAT0(BASE),(ATEMP)+
 231    0000013A 16EE FFFA           MOVE.B   STAT1(BASE),(ATEMP)+
 232    0000013E 16EE FFFB           MOVE.B   STAT2(BASE),(ATEMP)+
 233    00000142 16AE FFFC           MOVE.B   STAT3(BASE),(ATEMP)
 234    00000146 266E 000C           MOVEA.L  TRACKA(BASE),ATEMP
 235    0000014A 36C4                MOVE.W   DZERO,(ATEMP)+
 236    0000014C 16EE FFF5           MOVE.B   ADDR0(BASE),(ATEMP)+
 237    00000150 16AE FFF6           MOVE.B   ADDR1(BASE),(ATEMP)
 238    00000154 4E75        FINISH  RTS
 239                         *
 240                         *
 241                         *  DSJ ROUTINE
 242                         *
 243    00000156 1884        DSJR    MOVE.B   DZERO,(STATP)    DISABLE INTERRUPTS
 244    00000158 1544 0013           MOVE.B   DZERO,IMSK(PORT)
 245    0000015C 18BC 0040           MOVE.B   #ICMND,(STATP)
 246    00000160 12BC 003E           MOVE.B   #PLA,(FIFO)
 247    00000164 12AE FFFE           MOVE.B   DTA(BASE),(FIFO)
 248    00000168 12BC 0070           MOVE.B   #DSJ,(FIFO)
 249    0000016C 143C 0001           MOVE.B   #$01,BCOUNT              LOAD INPUT COUNT
 250    00000170 1884                MOVE.B   DZERO,(STATP)            INTERRUPT ENABLE
 251    00000172 157C 0004           MOVE.B   #$04,IMSK(PORT)          ENABLE "FIFO BYTE"
                 0013
 252    00000178 41EE FFFD           LEA      DSJBYTE(BASE),BYTEPTR     STORE ADDR. FOR DSJ
 253    0000017C 1884                MOVE.B   DZERO,(STATP)
 254    0000017E 12BC 0001           MOVE.B   #$01,(FIFO)              ENABLE TRANSFER
 255    00000182 4EBA 029A           JSR      IWAIT                    WAIT FOR INTERRUPT
 256    00000186 18BC 0040           MOVE.B   #ICMND,(STATP)
 257    0000018A 12BC 005F           MOVE.B   #UNT,(FIFO)
 258    0000018E 12BC 003F           MOVE.B   #UNL,(FIFO)
 259    00000192 6000 010C ENDSJR    BRA      IDLE
 260                         *
 261                         *
 262                         *  REQUEST AND READ DISK STATUS
 263                         *
 264    00000196 1544 0013 DSTAT     MOVE.B   DZERO,IMSK(PORT)
 265    0000019A 18BC 0040           MOVE.B   #ICMND,(STATP)
 266    0000019E 12BC 005E           MOVE.B   #PTA,(FIFO)
 267    000001A2 12AE FFFF           MOVE.B   DLA(BASE),(FIFO)
 268    000001A6 12BC 0068           MOVE.B   #DSAD,(FIFO)
 269    000001AA 6100 00DC           BSR      PNWAIT
 270    000001AE 1884                MOVE.B   DZERO,(STATP)
 271    000001B0 12BC 0003           MOVE.B   #$03,(FIFO)      OP CODE
 272    000001B4 18BC 0080           MOVE.B   #EOI,(STATP)
 273    000001B8 12AE 0025           MOVE.B   UNIT+1(BASE),(FIFO)
 274    000001BC 18BC 0040           MOVE.B   #ICMND,(STATP)
 275    000001C0 12BC 003F           MOVE.B   #UNL,(FIFO)
 276    000001C4 6100 00DA           BSR      IDLE
```



```
 277    000001C8 18BC 0040           MOVE.B   #ICMND,(STATP)
 278    000001CC 12BC 003E           MOVE.B   #PLA,(FIFO)
 279    000001D0 12AE FFFE           MOVE.B   DTA(BASE),(FIFO)
 280    000001D4 12BC 0068           MOVE.B   #DSAD,(FIFO)
 281    000001D8 143C 0004           MOVE.B   #$04,BCOUNT              LOAD INPUT COUNT
 282    000001DC 41EE FFF9           LEA      STAT0(BASE),BYTEPTR     STORE ADDR. FOR STAT
 283    000001E0 1884                MOVE.B   DZERO,(STATP)
 284    000001E2 157C 0004           MOVE.B   #$04,IMSK(PORT)    ENABLE "FIFO BYTE"
                 0013
 285    000001E8 18BC 0080           MOVE.B   #$80,(STATP)    LF INHIBIT
 286    000001EC 12BC 0004           MOVE.B   #$04,(FIFO)     COUNTED XFER ENABLE
 287    000001F0 4EBA 022C           JSR      IWAIT
 288    000001F4 18BC 0040           MOVE.B   #ICMND,(STATP)
 289    000001F8 12BC 005F           MOVE.B   #UNT,(FIFO)
 290    000001FC 12BC 003F           MOVE.B   #UNL,(FIFO)
 291    00000200 6000 009E           BRA      IDLE
 292                         *
 293                         *
 294                         *  REQUEST AND READ ADDRESS RECORD
 295                         *
 296    00000204 1544 0013 RADDREC   MOVE.B   DZERO,IMSK(PORT)
 297    00000208 18BC 0040           MOVE.B   #ICMND,(STATP)
 298    0000020C 12BC 005E           MOVE.B   #PTA,(FIFO)
 299    00000210 12AE FFFF           MOVE.B   DLA(BASE),(FIFO)
 300    00000214 12BC 0068           MOVE.B   #DSAD,(FIFO)
 301    00000218 6100 006E           BSR      PNWAIT
 302    0000021C 1884                MOVE.B   DZERO,(STATP)
 303    0000021E 12BC 0014           MOVE.B   #$14,(FIFO)      OP CODE
 304    00000222 18BC 0080           MOVE.B   #EOI,(STATP)
 305    00000226 12AE 0025           MOVE.B   UNIT+1(BASE),(FIFO)
 306    0000022A 18BC 0040           MOVE.B   #ICMND,(STATP)
 307    0000022E 12BC 003F           MOVE.B   #UNL,(FIFO)
 308    00000232 6100 006C           BSR      IDLE
 309    00000236 18BC 0040           MOVE.B   #ICMND,(STATP)
 310    0000023A 12BC 003E           MOVE.B   #PLA,(FIFO)
 311    0000023E 12AE FFFE           MOVE.B   DTA(BASE),(FIFO)
 312    00000242 12BC 0068           MOVE.B   #DSAD,(FIFO)
 313    00000246 143C 0004           MOVE.B   #$04,BCOUNT              LOAD INPUT COUNT
 314    0000024A 41EE FFF5           LEA      ADDR0(BASE),BYTEPTR     STORE ADDR. FOR ADDRESS REC
 315    0000024E 1884                MOVE.B   DZERO,(STATP)
 316    00000250 157C 0004           MOVE.B   #$04,IMSK(PORT)    ENABLE "FIFO BYTE"
                 0013
 317    00000256 18BC 0080           MOVE.B   #$80,(STATP)    LF INHIBIT
 318    0000025A 12BC 0004           MOVE.B   #$04,(FIFO)     COUNTED XFER ENABLE
 319    0000025E 4EBA 01BE           JSR      IWAIT
 320    00000262 18BC 0040           MOVE.B   #ICMND,(STATP)
 321    00000266 12BC 005F           MOVE.B   #UNT,(FIFO)
 322    0000026A 12BC 003F           MOVE.B   #UNL,(FIFO)
 323    0000026E 6000 0030           BRA      IDLE
 324                         *
 325                         *
 326                         * PPOLL HOLDOFF
 327                         *
 328    00000272 1544 001F PWAIT     MOVE.B   DZERO,PPSNS(PORT)          POSITIVE SENSE
 329    00000276 156E FFF4           MOVE.B   PMASK(BASE),PPMSK(PORT) SET PPMSK
                 001D
 330    0000027C 1884                MOVE.B   DZERO,(STATP)
```

```
PAGE   7 [2.0] 11/4/82 16:50:00  ASSEMBLY OF FORMAT.TEXT

 331    0000027E 157C 0020          MOVE.B  #$20,IMSK(PORT)          PPOLL RESPONSE
               0013
 332    00000284 6000 0198          BRA     IWAIT
 333                         *
 334                         *
 335                         *  PPOLL NEGATION HOLDOFF
 336                         *
 337    00000288 156E FFF4 PNWAIT   MOVE.B  PMASK(BASE),PPSNS(PORT) SET PPSNS
               001F
 338    0000028E 156E FFF4          MOVE.B  PMASK(BASE),PPMSK(PORT) SET PPMSK
               001D
 339    00000294 1884               MOVE.B  DZERO,(STATP)
 340    00000296 157C 0020          MOVE.B  #$20,IMSK(PORT)
               0013
 341    0000029C 6000 0180          BRA     IWAIT
 342                         *
 343                         *
 344                         *  FIFO IDLE HOLDOFF
 345                         *
 346    000002A0 1544 0013 IDLE     MOVE.B  DZERO,IMSK(PORT)
 347    000002A4 1884               MOVE.B  DZERO,(STATP)
 348    000002A6 157C 0002          MOVE.B  #$02,IMSK(PORT)    ENABLE "FIFO IDLE"
               0013
 349    000002AC 6000 0170          BRA     IWAIT
 350                         *
 351                         *
 352                         *  SEEK ROUTINE
 353                         *
 354    000002B0 18BC 0040 SEEK     MOVE.B  #ICMND,(STATP)
 355    000002B4 12BC 005E          MOVE.B  #PTA,(FIFO)
 356    000002B8 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
 357    000002BC 12BC 0068          MOVE.B  #DSAD,(FIFO)    SEEK SECONDARY
 358    000002C0 61C6               BSR     PNWAIT
 359    000002C2 1884               MOVE.B  DZERO,(STATP)
 360    000002C4 12BC 0002          MOVE.B  #$02,(FIFO)     OP CODE
 361    000002C8 12AE 0025          MOVE.B  UNIT+1(BASE),(FIFO)
 362    000002CC 12AE FFF0          MOVE.B  CYLINDR1(BASE),(FIFO)   CYLAD HIGH BYTE
 363    000002D0 12AE FFF1          MOVE.B  CYLINDR1+1(BASE),(FIFO)   CYLAD LOW BYTE
 364    000002D4 12AE 0023          MOVE.B  HEAD+1(BASE),(FIFO)
 365    000002D8 61C6               BSR     IDLE            EMPTY FIFO
 366    000002DA 18BC 0080          MOVE.B  #EOI,(STATP)
 367    000002DE 12BC 0000          MOVE.B  #SECTOR,(FIFO)
 368    000002E2 18BC 0040          MOVE.B  #ICMND,(STATP)
 369    000002E6 12BC 003F          MOVE.B  #UNL,(FIFO)
 370    000002EA 61B4               BSR     IDLE
 371    000002EC 6084               BRA     PWAIT     WAIT FOR SEEK COMPLETION
 372                         *
 373                         *
 374                         *  INITIALIZE
 375                         *
 376    000002EE 18BC 0040 INITZE   MOVE.B  #ICMND,(STATP)
 377    000002F2 12BC 005E          MOVE.B  #PTA,(FIFO)
 378    000002F6 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
 379    000002FA 12BC 0068          MOVE.B  #DSAD,(FIFO)
 380    000002FE 6188               BSR     PNWAIT
 381    00000300 102E FFF2          MOVE.B  SPDTEMP(BASE),DTEMP
 382    00000304 0600 000B          ADDI.B  #$B,DTEMP       ADD SPD TO OP CODE
```

```
PAGE   8 [2.0] 11/4/82 16:50:00  ASSEMBLY OF FORMAT.TEXT

 383    00000308 1280               MOVE.B  DTEMP,(FIFO)
 384    0000030A 18BC 0080          MOVE.B  #EOI,(STATP)
 385    0000030E 12AE 0025          MOVE.B  UNIT+1(BASE),(FIFO)
 386    00000312 18BC 0040          MOVE.B  #ICMND,(STATP)
 387    00000316 12BC 003F          MOVE.B  #UNL,(FIFO)
 388    0000031A 6184               BSR     IDLE
 389
 390    0000031C 18BC 0040          MOVE.B  #ICMND,(STATP)
 391    00000320 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
 392    00000324 12BC 0060          MOVE.B  #$60,(FIFO)     SEC. WDAT
 393
 394    00000328 1884               MOVE.B  DZERO,(STATP)
 395    0000032A 1D44 FFF3          MOVE.B  DZERO,IFLAG(BASE)
 396    0000032E 33FC 000C          MOVE.W  #$0C,DMCTRL0    (PRIORITY,OUTPUT, BYTE)
               0050 0006
 397    00000336 157C 0089          MOVE.B  #$89,CRDCTRL(PORT) (INTERRUPT,OUTPUT,CHAN. 0)
               0003
 398    0000033C 157C 0002          MOVE.B  #$02,CTRL(PORT)   ENABLE DMARQ, 10 BIT PROC.?
               0019
 399                         * NOW DATA IS BEING TRANSFERRED
 400    00000342 263C 0005          MOVE.L  #363636,TCOUNT    2 SEC. TIMEOUT
               8C74
 401    00000348 4A2E FFF3 LOOPW    TST.B   IFLAG(BASE)
 402    0000034C 6608               BNE.S   WDONE
 403    0000034E 5383               SUBQ.L  #1,TCOUNT
 404    00000350 6CF6               BGE     LOOPW
 405                         * FAILED TO DO IT IN 2 SECONDS
 406    00000352 6000 0106          BRA     TESCAPE
 407    00000356 157C 0080 WDONE    MOVE.B  #$80,CTRL(PORT)
               0019
 408    0000035C 18BC 0040          MOVE.B  #ICMND,(STATP)
 409    00000360 12BC 003F          MOVE.B  #UNL,(FIFO)
 410    00000364 12BC 005F          MOVE.B  #UNT,(FIFO)
 411    00000368 6000 FF36          BRA     IDLE
 412                         *
 413                         *
 414                         *  ADDRESS RECORD ROUTINE
 415                         *
 416    0000036C 18BC 0040 ADDREC   MOVE.B  #ICMND,(STATP)
 417    00000370 12BC 005E          MOVE.B  #PTA,(FIFO)
 418    00000374 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
 419    00000378 12BC 0068          MOVE.B  #DSAD,(FIFO)    ADDRESS RECORD SECONDARY
 420    0000037C 6100 FF0A          BSR     PNWAIT
 421    00000380 1884               MOVE.B  DZERO,(STATP)
 422    00000382 12BC 000C          MOVE.B  #$0C,(FIFO)     OP CODE
 423    00000386 12AE 0025          MOVE.B  UNIT+1(BASE),(FIFO)
 424    0000038A 12AE FFEE          MOVE.B  CYLINDR2(BASE),(FIFO)   CYLAD HIGH BYTE
 425    0000038E 12AE FFEF          MOVE.B  CYLINDR2+1(BASE),(FIFO)   CYLAD LOW BYTE
 426    00000392 12AE 0023          MOVE.B  HEAD+1(BASE),(FIFO)
 427    00000396 6100 FF08          BSR     IDLE            EMPTY FIFO
 428    0000039A 18BC 0080          MOVE.B  #EOI,(STATP)
 429    0000039E 12BC 0000          MOVE.B  #SECTOR,(FIFO)
 430    000003A2 18BC 0040          MOVE.B  #ICMND,(STATP)
 431    000003A6 12BC 003F          MOVE.B  #UNL,(FIFO)
 432    000003AA 6000 FEC6          BRA     PWAIT     WAIT FOR COMPLETION
 433                         *
 434                         *
```

```
                        *  EMPTY INBOUND FIFO
                        *
000003AE 1884           INEMP   MOVE.B  DZERO,(STATP)
000003B0 157C 0004              MOVE.B  #$04,IMSK(PORT)
         0013
000003B6 4A2A 0011 EMPTY   TST.B   INTR(PORT)              CHECK FOR BYTE AVAILABLE
000003BA 6700 0006         BEQ     EMPTY1                IF SO, READ, IF NOT RETURN.
000003BE 4A11              TST.B   (FIFO)                (DUMMY READ OF FIFO TO EMPTY IT)
000003C0 60F4              BRA     EMPTY
000003C2 4E75           EMPTY1  RTS
                        *
                        *
                        *  SET FILE MASK ROUTINE
                        *     OUTPUT BYTE =XXXXDSCA
                        *     D=DECREMENTAL SEEK (1) OR INCREMENTAL SEEK(0)
                        *     S=1 ALLOWS AUTOMATIC SEEK TO SPARE TRACK
                        *     C=1 ENABLES CYLINDER MODE, C=0 SURFACE MODE
                        *     A=1 ENABLES AUTOMATIC SEEK AND END OF CYLINDER (SEE D)
                        *
                        *
000003C4 18BC 0040 SFM     MOVE.B  #ICMND,(STATP)
000003C8 12BC 005E         MOVE.B  #PTA,(FIFO)
000003CC 12AE FFFF         MOVE.B  DLA(BASE),(FIFO)
000003D0 12BC 0068         MOVE.B  #DSAD,(FIFO)
000003D4 6100 FEB2         BSR     PWAIT
000003D8 1884              MOVE.B  DZERO,(STATP)
000003DA 12BC 000F         MOVE.B  #$0F,(FIFO)
000003DE 18BC 0080         MOVE.B  #E0I,(STATP)
000003E2 12BC 0001         MOVE.B  #1,(FIFO)
                        *   DISABLE AUTO SEEK TO SPARE,EN SURFACE, AUTO
                        *     SEEK TO NEXT CYLINDER
000003E6 18BC 0040         MOVE.B  #ICMND,(STATP)
000003EA 12BC 003F         MOVE.B  #UNL,(FIFO)
000003EE 6100 FEB0         BSR IDLE
000003F2 6000 FE7E         BRA PWAIT
                        *
                        *
                        *  INTERRUPT SERVICE ROUTINE
                        *
                        *
                        *  SERVICE EOP INTERRUPT
                        *
000003F6 265F           ISV     MOVEA.L (SP)+,ATEMP           RETURN ADDRESS
000003F8 2C5F                   MOVEA.L (SP)+,BASE            STATIC LINK
000003FA 2E8B                   MOVE.L  ATEMP,(SP)            (POP PARAMETER & PUSH) RETURN ADDRESS
000003FC 246E 001E              MOVEA.L PHIPORT(BASE),PORT
00000400 43EA 0015              LEA     FIFOFFSET(PORT),FIFO
00000404 49EA 0017              LEA     STATPOFFS(PORT),STATP
00000408 4284                   CLR.L   DZERO
                        *
0000040A 1544 0003              MOVE.B  DZERO,CRDCTRL(PORT)   DISABLE INTERRUPTS,DMA
0000040E 4A2A 0019              TST.B   CTRL(PORT)            DUMMY READ TO CLEAR INTERRUPT
00000412 6100 FE5E              BSR     PWAIT
00000416 1D7C 0001              MOVE.B  #$01,IFLAG(BASE)
         FFF3
0000041C 4E75                   RTS
```



```
                        *
                        *
                        *  IWAIT ROUTINE
0000041E 363C E77A IWAIT   MOVE.W  #59258,TCOUNT         TIME OUT COUNT FOR 200 MILLISECONDS
00000422 102A 0011 IIWAIT  MOVE.B  INTR(PORT),DTEMP
00000426 56CB FFFA         DBNE    TCOUNT,IIWAIT         TRY AGAIN
0000042A 672E              BEQ.S   TESCAPE               TIMED OUT IF STILL CLEAR
0000042C B03C 0004         CMP.B   #04,DTEMP
00000430 6700 000E         BEQ     BYTE                  CHECK INTERRUPT TYPE
00000434 B03C 0020         CMP.B   #$20,DTEMP
00000438 6700 001E         BEQ     PPW
0000043C 6000 0014         BRA     OUT
                        *
                        *  SERVICE FIFO BYTE AVAILABLE INTERRUPT
                        *
00000440 10D1           BYTE    MOVE.B  (FIFO),(BYTEPTR)+     STORE INPUT DATA
00000442 5302                   SUBQ.B  #1,BCOUNT                 DECREMENT INPUT COUNT
00000444 6700 000C              BEQ     OUT
00000448 0C2A 0004 TEST    CMPI.B  #4,INTR(PORT)         ANOTHER BYTE?
         0011
0000044E 66F8              BNE     TEST
00000450 60EE              BRA     BYTE
00000452 1884           OUT     MOVE.B  DZERO,(STATP)         DISABLE INTERRUPTS
00000454 1544 0013              MOVE.B  DZERO,IMSK(PORT)
                        *
                        *
                        *  SERVICE PPOLL INTERRUPT
                        *
00000458 4E75           PPW     RTS
                        *
0000045A 6100 000E TESCAPE BSR     HDWRCLEAR          INVOKE IFC, CLEAR DMA, ETC.
0000045E 3B7C FFF6         MOVE.W  #-10,SYSGLOBALS-2(GLOBAL)     ESCAPE(-10)
         FFFE
00000464 2E6D FFF6         MOVEA.L SYSGLOBALS-10(GLOBAL),SP
00000468 4E75              RTS
                        *
                        *
                        * HARDWARE CLEAR
                        *
            0000 046A HDWRCLEAR EQU *
0000046A 4A79 0050         TST.W   DMSTAT                READ DMA CARD CONTROL REG TO DISARM IT
         0000
00000470 1544 0001         MOVE.B  DZERO,ID(PORT)           SOFTWARE RESET
00000474 157C 0080         MOVE.B  #$80,HPIBADR(PORT)    ON-LINE
         001B
0000047A 6100 0010         BSR     IFCLEAR               ASSERT IFC
                        * DISK CLEAR FOR HANGUPS
0000047E 157C 0040         MOVE.B  #ICMND,STATPOFFS(PORT)
         0017
00000484 157C 0014         MOVE.B  #$14,FIFOFFSET(PORT)  DEVICE CLEAR
         0015
0000048A 4E75              RTS
                        *
                        *
                        * INTERFACE CLEAR
                        *
```

```
 541    0000048C 157C 0011 IFCLEAR MOVE.B  #$11,CTRL(PORT)        IFC, INIT OUTBOUND FIFO
                  0019
 542    00000492 7642      DEL100  MOVEQ   #66,TCOUNT             100 MICROSECOND DELAY
 543    00000494 51CB FFFE REP     DBRA    TCOUNT,REP
 544    00000498 157C 0080         MOVE.B  #$80,CTRL(PORT)        8-BIT PROCESSOR
                  0019
 545    0000049E 4E75              RTS
 546                      *
 547                      *
 548                      *
 549                        END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

GPIO

Purpose

GPIO contains assembly language low-level drivers.

Usage

GPIO is used for the 98622 interface.

Requirements

G_DRV and COMASM

PASS 1 COMPLETE. ERRORS: 0


```
                        ********************************************************************************
                        *
                        *       COPYRIGHT (C) 1982 BY HEWLETT-PACKARD COMPANY
                        *
                        ********************************************************************************
                        *
                        *
                        *       IOLIB     EXTG
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *
                        *       Library - IOLIB
                        *
                        *       Purpose - This set of assembly language code is intended to be used as
                        *                 a PASCAL module for I/O drivers for use by the external I/O
                        *                 procedures library.
                        *
                        *
                        *       Date    - 09/20/81
                        *       Update  - 10/06/82
                        *       Release - 10/06/82
                        *
                        *
                        *       Source  - IOLIB:GPIO.TEXT
                        *       Object  - IOLIB:GPIO.CODE
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *       RELEASED
                        *       VERSION         2.0
                        *
                        *
                        ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *       The following lines are used to tell the LINKER/LOADER what this module
                        *       looks like in PASCAL terms.
                        *
                        *       Note that it is possible to create assembly modules that are functions.
                        *       These routines are called through an indirect pointer using the CALL
                        *       facility which does NOT permit functions.
                        *
                        *       This module is called 'EXTG' ( upper or lower case - doesn't matter )
                        *       independent of the file name ( by use of the MNAME pseudo-op ).
                        *
                        *       All the externally used procedures are called 'EXTG_@@@@@@@' in
                        *       this module.  If you are using assembly to access them use the
                        *       'EXTG_@@@@@@@' name.  If you are using Pascal use the '@@@@@@@'
                        *       name.
                        *
                        ********************************************************************************
                                MNAME EXTG
                                SRC MODULE EXTG;
                                SRC IMPORT iodeclarations;
                                SRC EXPORT
                                SRC         PROCEDURE eg_init   ( temp : ANYPTR );
                                SRC         PROCEDURE eg_isr    ( temp : ANYPTR );
                                SRC         PROCEDURE eg_rdb    ( temp : ANYPTR ;  VAR x : CHAR);
                                SRC         PROCEDURE eg_wtb    ( temp : ANYPTR ;  val   : CHAR);
                                SRC         PROCEDURE eg_rdw    ( temp : ANYPTR ;  VAR x : io_word);
                                SRC         PROCEDURE eg_wtw    ( temp : ANYPTR ;  val   : io_word);
                                SRC         PROCEDURE eg_rds    ( temp : ANYPTR ;  reg   : io_word;
                                SRC                                                VAR x : io_word);
                                SRC         PROCEDURE eg_wtc    ( temp : ANYPTR ;  reg   : io_word;
                                SRC                                                val   : io_word );
                                SRC         PROCEDURE eg_tfr    ( temp : ANYPTR ;  bcb   : ANYPTR );
                                SRC         PROCEDURE eg_clr    ( temp : ANYPTR ;  line  : io_bit );
                                SRC         PROCEDURE eg_set    ( temp : ANYPTR ;  line  : io_bit );
                                SRC         PROCEDURE eg_test   ( temp : ANYPTR ;  line  : io_bit ;
                                SRC                                                VAR x : BOOLEAN );
                                SRC END; ( of EXTG )
```



```
                        ********************************************************************************
                        *
                        *       SYMBOLS FOR EXPORT AS PROCEDURE NAMES
                        *
                        ********************************************************************************
                                DEF EXTG_EXTG

                                DEF EXTG_EG_INIT
                                DEF EXTG_EG_ISR,EXTG_EG_TDMA
                                DEF EXTG_EG_RDB,EXTG_EG_WTB
                                DEF EXTG_EG_RDW,EXTG_EG_WTW
                                DEF EXTG_EG_RDS,EXTG_EG_WTC
                                DEF EXTG_EG_TFR

                                DEF EXTG_EG_SET,EXTG_EG_CLR,EXTG_EG_TEST

                        ********************************************************************************
                        *
                        *       SYMBOLS FOR IMPORT
                        *
                        ********************************************************************************
                                REFA STBSY
                                REFA STCLR
                                REFA ITXFR
                                REFA ABORT_IO
                                REFA LOGINT
                                REFA GETDMA
                                REFA DROPDMA
                                REFA TESTDMA
                                REFA DMA_STBSY
                                REFA WAIT_TFR
                                REFA CHECK_TFR
                        *       change references to allow long jumps when the I/O      475 TM 9/17/82
                        *       modules get moved around                                475 TM 9/17/82
                                LMODE STBSY                                             475 TM 9/17/82
                                LMODE STCLR                                             475 TM 9/17/82
                                LMODE ITXFR                                             475 TM 9/17/82
                                LMODE ABORT_IO                                          475 TM 9/17/82
                                LMODE LOGINT                                            475 TM 9/17/82
                                LMODE GETDMA                                            475 TM 9/17/82
                                LMODE DROPDMA                                           475 TM 9/17/82
                                LMODE TESTDMA                                           475 TM 9/17/82
                                LMODE DMA_STBSY                                         475 TM 9/17/82
                                LMODE WAIT_TFR                                          475 TM 9/17/82
                                LMODE CHECK_TFR                                         475 TM 9/17/82
```

```
PAGE   5 [2.0] 11/4/82 16:22:45 IOLIB EXTG - COMMON DECLARATIONS AND EQUATES
 130                            INCLUDE COMDCL
```

```
PAGE   6 [2.0] 11/4/82 16:22:45 IOLIB COMDCL - COMMON EQUATES AND DEFINITIONS
 133                    ********************************************************************************
 134                    *
 135                    *       modified:  02/22/82  JPC   added parm to user EOT & ISR proc's
 136                    *
 137                    ********************************************************************************
 138                    *
 139                    *       HPL CONVENTIONS
 140                    *
 141                    *
 142                    *         Much of this code is taken intact from the 9826 HPL
 143                    *         language system EIO ROM ( extended I/O ROM ).
 144                    *
 145                    *         The Pascal that will be calling this code uses
 146                    *         the stack for parameter passage.  The HPL code
 147                    *         uses the Ax and Dx registers for all parameters.
 148                    *         The Pascal driver entry points on the previous pages
 149                    *         take care of getting the parameters into the correct
 150                    *         registers.
 151                    *
 152                    *
 153                    *       GENERAL HPL ENTRY/EXIT CONDITIONS:
 154                    *
 155                    *         A1.L = CARD ADDRESS
 156                    *         A2.L = DRIVER TEMP ADDRESS
 157                    *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
 158                    *
 159                    *
 160                    *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
 161                    *
 162                    *         A3.L = BUFFER CONTROL BLOCK ADDRESS
 163                    *                In addition to the A1/A2 convention, Pascal will use
 164                    *                A3 for a pointer to the buffer control block.
 165                    *                The HPL system kept much of the transfer
 166                    *                information in the s.c. temps.
 167                    *
 168                    *         TIMEOUT(A2) = contains timeout information
 169                    *                Timeout was a global temp in HPL and a timeout
 170                    *                generated an error.
 171                    *                In PASCAL each card has a timeout value stored in
 172                    *                its temporary area.  A timeout error
 173                    *                generates an ESCAPE ( which can be trapped ).
 174                    *
 175                    *
 176                    ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *         DRIVER TEMPS TEMPLATE
                        *
                        *           OFFSET FROM A2
                        *
                        *           HPL DECLARATIONS ( MODIFIED )
                        *
                        *
                        ********************************************************************************
0000 0000 ISR_ENTRY     EQU 0       ..19        PASCAL ISR LINK & UNLINK area
0000 0014 USER_ISR      EQU 20                  user ISR:  do NOT change the proc/stat link/parm ordering!!!
0000 0014 H_ISR_PR      EQU 20      ..23            procedure ptr
0000 0018 H_ISR_SL      EQU 24      ..27            static link
0000 001C H_ISR_PM      EQU 28      ..31            parameter
0000 0020 C_ADR         EQU 32      ..35        card address
0000 0024 BUFI_OFF      EQU 36      ..39        buffer pointer offset
0000 0028 BUFO_OFF      EQU 40      ..43        buffer pointer offset
0000 002C EIRB_OFF      EQU 44                  eir byte
0000 002D IO_SC         EQU 45                  select code ( i.e. 7, 22, etc. )
0000 002E TIMEOUT       EQU 46      ..49        timeout value
                        *                           =0 : no timeout
                        *                           #0 : value of timeout
0000 0032 MA_W          EQU 50      ..51        word access to my address
0000 0033 MA            EQU 51                  byte access to my address
0000 0034 AVAIL_OFF     EQU 52      ..??        standard space taken from temps
                        *               52      ..83        normal cards ( 32  bytes )
                        *               52      ..179       98628  card  ( 128 bytes )
```



```
                        ********************************************************************************
                        *
                        *       TRANSFER OFFSETS IN BUFFER CONTROL BLOCK
                        *
                        *           OFFSET FROM A3
                        *
                        *           PASCAL DECLARATION
                        *
                        ********************************************************************************
0000 0000 TTMP_OFF      EQU 0   ..3         pointer to driver temp offset
0000 0005 T_SC_OFF      EQU 5               transfer select code
0000 0007 TACT_OFF      EQU 7               actual transfer mode
0000 0009 TUSR_OFF      EQU 9               transfer mode
                        *                       00   -                  not used
                        *                       01   serial DMA
                        *                       02   serial FHS
                        *                       03   serial FASTEST ( DMA or FHS )
                        *                       04   -                  not used
                        *                       ----------------
                        *                       05   overlp INTR
                        *                       06   overlp DMA
                        *                       07   overlp FHS      ( BURST )
                        *                       08   overlp FASTEST ( DMA or BURST )
                        *                       09   overlp OVERLAP ( DMA or INTR )
0000 000A T_BW_OFF      EQU 10              transfer byte/word indicator
                        *                       0 = byte   / 1 = word
0000 000B TEND_OFF      EQU 11              transfer EOI/END indicator
                        *                       0 = no eoi / 1 = eoi sent or searched for
0000 000D TDIR_OFF      EQU 13              transfer direction
                        *                       0 = input  /  1 = output
0000 000E TCHR_OFF      EQU 14  ..15        transfer terminate character
                        *                       -1 = no termination character
0000 0010 TCNT_OFF      EQU 16  ..19        transfer count
0000 0014 TBUF_OFF      EQU 20  ..23        transfer buffer pointer
0000 0018 TBSZ_OFF      EQU 24  ..27        transfer buffer maximum size
0000 001C TEMP_OFF      EQU 28  ..31        transfer empty pointer pointer
0000 0020 TFIL_OFF      EQU 32  ..35        transfer fill pointer
0000 0024 T_PR_OFF      EQU 36  ..39        transfer pointer to eot procedure
                        *                   NIL  no procedure
0000 0028 T_SL_OFF      EQU 40  ..43        transfer eot proc static link
0000 002C T_PM_OFF      EQU 44  ..47        transfer eot proc parameter
0000 0030 T_DMAPRI      EQU 48              dma priority request
                        *
                        *       TRANSFER EQUATES
                        *
0000 0001 TT_INT        EQU 1               interrupt
0000 0002 TT_DMA        EQU 2               DMA
0000 0003 TT_BURST      EQU 3               burst
0000 0004 TT_FHS        EQU 4               fast handshake
```

```
                        ********************************************************************************
                        *
                        *       EXTERNAL REFERANCES for escape
                        *
                        ********************************************************************************
                                REFA iodeclarations          reference the io lib var. area
                                REFA sysglobals


                        ********************************************************************************
                        *
                        *       Escape code values
                        *
                        ********************************************************************************
            0000 0001 NO_CARD   EQU    1                     no interface
            0000 0002 NOT_HPIB  EQU    2                     not an hpib interface
            0000 0003 NO_ACTL   EQU    3                     no active controller
            0000 0004 NO_DVC    EQU    4                     sc ( not device ) specified
            0000 0005 NO_SPACE  EQU    5                     not enough space in the buffer
            0000 0006 NO_DATA   EQU    6                     not enough data left in the buffer
            0000 0007 TFR_ERR   EQU    7                     tfr error
            0000 0008 SC_BUSY   EQU    8                     sc is currently busy
            0000 0009 BUF_BUSY  EQU    9                     the buffer is busy
            0000 000A TCNTERR   EQU    10                    bad count
            0000 000B BADTMO    EQU    11                    bad timeout value
            0000 000C NO_DRV    EQU    12                    no driver
            0000 000D NO_DMA    EQU    13                    no dma installed
            0000 000E NO_WORD   EQU    14                    no word transfers allowed
            0000 000F NOT_TALK  EQU    15                    not addressed as talker
            0000 0010 NOT_LSTN  EQU    16                    not addressed as listener
            0000 0011 TMO_ERR   EQU    17                    timeout
            0000 0012 NO_SCTL   EQU    18                    not system controller
            0000 0013 BAD_RDS   EQU    19                    bad read status / write control
            0000 0014 BAD_SCT   EQU    20                    bad set/clear/test
            0000 0015 CRD_DWN   EQU    21                    interface is dead
            0000 0016 EOD_SEEN  EQU    22                    end of data has happened
            0000 0017 IO_MISC   EQU    23                    misc. error


            FFFF FFE6 IOE_ERROR EQU    -26                   io sub system error escape code


            FFFF FFBE IOE_RSLT  EQU    IODECLARATIONS-66
            FFFF FFBA IOE_SC    EQU    IODECLARATIONS-70


            FFFF FFFE ESC_CODE  EQU    SYSGLOBALS-2
            FFFF FFF6 RCVR_BLK  EQU    SYSGLOBALS-10

                        ********************************************************************************
                        *
                        *         PASCAL DRIVER ENTRY POINTS FOR GPIO CARDS
                        *
                        ********************************************************************************
```



```
                        *
                        *         MODULE initialization
                        *
            0000 0000 EXTG_EXTG EQU *
00000000 4E75               RTS
                        *
                        *         Driver initialization
                        *
            0000 0002 EXTG_EG_INIT      EQU *
00000002 205F               MOVEA.L (SP)+,A0      get return address
00000004 245F               MOVEA.L (SP)+,A2      get temp address
00000006 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
0000000A 4850               PEA     (A0)          push return address back on stack
0000000C 6000 00D4          BRA     G_INIT
                        *
                        *         Interrupt service routine
                        *
            0000 0010 EXTG_EG_ISR       EQU *
00000010 205F               MOVEA.L (SP)+,A0      get return address
00000012 245F               MOVEA.L (SP)+,A2      get temp address
00000014 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
00000018 4850               PEA     (A0)          push return address back on stack
0000001A 6000 04F8          BRA     G_ISR
                        *
                        *         HPIB DMA transfer termination routine
                        *
            0000 001E EXTG_EG_TDMA      EQU *
0000001E 205F               MOVEA.L (SP)+,A0      get return address
00000020 245F               MOVEA.L (SP)+,A2      get temp address
00000022 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
00000026 4850               PEA     (A0)          push return address back on stack
00000028 6000 0480          BRA     G_DMATERM
                        *
                        *         Read a byte
                        *
            0000 002C EXTG_EG_RDB       EQU *
0000002C 205F               MOVEA.L (SP)+,A0      get return address
0000002E 265F               MOVEA.L (SP)+,A3      get VAR address
00000030 245F               MOVEA.L (SP)+,A2      get temp address
00000032 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
00000036 4850               PEA     (A0)          push return address back on stack
00000038 6100 00E8          BSR     G_RDB         call read byte ( or word )
0000003C 1680               MOVE.B  D0,(A3)       save character
0000003E 4E75               RTS
                        *
                        *         Write a byte
                        *
            0000 0040 EXTG_EG_WTB       EQU *
00000040 205F               MOVEA.L (SP)+,A0      get return address
00000042 101F               MOVE.B  (SP)+,D0      get value          ( this actually bumps SP by 2 )
00000044 245F               MOVEA.L (SP)+,A2      get temp address
00000046 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
0000004A 4850               PEA     (A0)          push return address back on stack
0000004C 6000 00C8          BRA     G_WTB         call write byte ( or word )
                        *
                        *         Read a word
```

```
                        *
            0000 0050 EXTG_EG_RDW       EQU *
00000050 205F                   MOVEA.L (SP)+,A0        get return address
00000052 265F                   MOVEA.L (SP)+,A3        get VAR address
00000054 245F                   MOVEA.L (SP)+,A2        get temp address
00000056 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
0000005A 4850                   PEA     (A0)            push return address back on stack
0000005C 6100 00C4              BSR     G_RDB           call read byte ( or word )
00000060 3680                   MOVE.W  D0,(A3)         save word
00000062 4E75                   RTS
                        *
                        *       Write a word
                        *
            0000 0064 EXTG_EG_WTW       EQU *
00000064 205F                   MOVEA.L (SP)+,A0        get return address
00000066 301F                   MOVE.W  (SP)+,D0        get word value
00000068 245F                   MOVEA.L (SP)+,A2        get temp address
0000006A 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
0000006E 4850                   PEA     (A0)            push return address back on stack
00000070 6000 00A4              BRA     G_WTB           write the byte ( or word )
                        *
                        *       Read status
                        *
            0000 0074 EXTG_EG_RDS       EQU *
00000074 205F                   MOVEA.L (SP)+,A0        get return address
00000076 265F                   MOVEA.L (SP)+,A3        get VAR address
00000078 321F                   MOVE.W  (SP)+,D1        get register number
0000007A 245F                   MOVEA.L (SP)+,A2        get temp address
0000007C 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
00000080 4850                   PEA     (A0)            push return address back on stack
00000082 6100 0192              BSR     G_RDS           get status
00000086 3680                   MOVE.W  D0,(A3)         save status info
00000088 4E75                   RTS
                        *
                        *       Write control
                        *
            0000 008A EXTG_EG_WTC       EQU *
0000008A 205F                   MOVEA.L (SP)+,A0        get return address
0000008C 301F                   MOVE.W  (SP)+,D0        get value
0000008E 321F                   MOVE.W  (SP)+,D1        get register number
00000090 245F                   MOVEA.L (SP)+,A2        get temp address
00000092 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
00000096 4850                   PEA     (A0)            push return address back on stack
00000098 6000 01D8              BRA     G_WTC           write control
                        *
                        *       Transfer
                        *
            0000 009C EXTG_EG_TFR       EQU *
0000009C 205F                   MOVEA.L (SP)+,A0        get return address
0000009E 265F                   MOVEA.L (SP)+,A3        get buffer control block address
000000A0 245F                   MOVEA.L (SP)+,A2        get temp address
000000A2 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
000000A6 4850                   PEA     (A0)            push return address back on stack
000000A8 6000 022C              BRA     G_TFR           transfer
                        *
                        *       Set an GPIO line
                        *
```



```
            0000 00AC EXTG_EG_SET       EQU *
000000AC 205F                   MOVEA.L (SP)+,A0        get return address
000000AE 321F                   MOVE.W  (SP)+,D1        get line            ( this actually bumps SP by 2 )
000000B0 245F                   MOVEA.L (SP)+,A2        get temp address
000000B2 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
000000B6 4850                   PEA     (A0)            push return address back on stack
000000B8 6000 00E6              BRA     G_SET           call set line
                        *
                        *       Clear an GPIO line
                        *
            0000 00BC EXTG_EG_CLR       EQU *
000000BC 205F                   MOVEA.L (SP)+,A0        get return address
000000BE 321F                   MOVE.W  (SP)+,D1        get line            ( this actually bumps SP by 2 )
000000C0 245F                   MOVEA.L (SP)+,A2        get temp address
000000C2 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
000000C6 4850                   PEA     (A0)            push return address back on stack
000000C8 6000 00FA              BRA     G_CLR           clear the line
                        *
                        *       Test an GPIO line
                        *
            0000 00CC EXTG_EG_TEST      EQU *
000000CC 205F                   MOVEA.L (SP)+,A0        get return address
000000CE 265F                   MOVEA.L (SP)+,A3        get VAR address
000000D0 321F                   MOVE.W  (SP)+,D1        get line            ( this actually bumps SP by 2 )
000000D2 245F                   MOVEA.L (SP)+,A2        get temp address
000000D4 226A 0020              MOVEA.L C_ADR(A2),A1    get card address
000000D8 4850                   PEA     (A0)            push return address back on stack
000000DA 6100 0106              BSR     G_TEST          read status
000000DE 1680                   MOVE.B  D0,(A3)         save character
000000E0 4E75                   RTS
```

```
                        ****************************************
                        *
                        *  GPIO INITIALIZATION
                        *
                        ****************************************
            0000 00E2 G_INIT   EQU *
000000E2 4EB9 0000 G_RESET  JSR     ABORT_IO                                        475 TM 9/17/82
                0000
000000E8 357C FFFF          MOVE.W  #-1,MA_W(A2)   the card is not addressable - so my addr =-1
                0032
000000EE 1340 0001          MOVE.B  D0,1(A1)       Reset the GPIO card
000000F2 7413               MOVEQ   #19,D2         Wait 15 microseconds
000000F4 51CA FFFE          DBRA    D2,*
000000F8 137C 0003          MOVE.B  #3,2(A1)
                0002
000000FE 4E75               RTS
                        ****************************************
                        *
                        *  IOFS
                        *
                        ****************************************
00000100 7000       G_IOFS  MOVEQ   #0,D0
00000102 0829 0003          BTST    #3,7(A1)    PSTS -> D0<0>
                0007
00000108 6702               BEQ.S   G_IOFS1
0000010A 7001               MOVEQ   #1,D0
0000010C 0811 0000 G_IOFS1  BTST    #0,(A1)     PFLG -> D0<1>
00000110 6702               BEQ.S   G_IOFS2
00000112 5440               ADDQ    #2,D0
00000114 4E75       G_IOFS2 RTS
```



```
                        ****************************************
                        *
                        *  wtb
                        *
                        ****************************************
00000116 611C       G_WTB   BSR.S   G_STSCHK     Check for PSTS
00000118 6134               BSR.S   G_WAIT       Wait for FLG
0000011A 3340 0004          MOVE.W  D0,4(A1)     Write the word
0000011E 1280               MOVE.B  D0,(A1)      wti 7,0  trigger output
00000120 4E75               RTS
                        ****************************************
                        *
                        *  rdb
                        *
                        ****************************************
00000122 6110       G_RDB   BSR.S   G_STSCHK     Check for PSTS
00000124 6128               BSR.S   G_WAIT
00000126 3029 0004          MOVE.W  4(A1),D0     Set I/O line for input
0000012A 1280               MOVE.B  D0,(A1)      wti 7,0  trigger input
0000012C 6120               BSR.S   G_WAIT       Wait for data to come in
0000012E 3029 0004          MOVE.W  4(A1),D0
00000132 4E75               RTS
00000134 3F00       G_STSCHK MOVE   D0,-(SP)     Save D0
00000136 303C 55F0          MOVE    #22000,D0    Count parameter for 100 ms wait
0000013A 0829 0003 G_STS0   BTST    #3,7(A1)     Is PSTS O.K. ?
                0007
00000140 6704               BEQ.S   G_STS1       No:  Give him 100 ms
00000142 301F               MOVE    (SP)+,D0     Yes: Restore D0
00000144 4E75               RTS
00000146 51C8 FFF2 G_STS1   DBRA    D0,G_STS0    Decrement loop counter
0000014A 6000 0030          BRA     CRD_DOWN     If counter expires, card is down
```

```
                        ****************************************
                        *
                        *  GPIO WAIT ROUTINE
                        *
                        ****************************************
0000014E 0811 0000 G_WAIT   BTST    #0,(A1)         IF CARD_READY THEN RETURN
00000152 6622               BNE.S   G_WRTS
00000154 222A 002E          MOVE.L  TIMEOUT(A2),D1 D1 = (TIMEOUT)
00000158 6716               BEQ.S   G_WAIT4
            0000 015A G_WAIT2  EQU     *
                        * tm     MULU    #182,D1         CONVERT D1 TO MILLISECONDS
0000015A 2A01               MOVE.L  D1,D5           \
0000015C EF89               LSL.L   #7,D1            \  * 196 IS CLOSE TO * 182
0000015E ED8D               LSL.L   #6,D5            /
00000160 D285               ADD.L   D5,D1           /
00000162 0811 0000 G_WAIT3  BTST    #0,(A1)         WHILE D1 MILLISECONDS HAS NOT EXPIRED DO
00000166 660E               BNE.S   G_WRTS          IF CARD_READY THEN RETURN
00000168 5381               SUBQ.L  #1,D1
0000016A 6EF6               BGT.S   G_WAIT3
0000016C 7011               MOVEQ   #TMO_ERR,D0     GENERATE ESCAPE
0000016E 601A               BRA.S   ESC_ERR


00000170 0811 0000 G_WAIT4  BTST    #0,(A1)         IF CARD_READY THEN RETURN
00000174 67FA               BEQ.S   G_WAIT4
00000176 4E75       G_WRTS  RTS
```



```
                        ********************************************************************************
                        *
                        *       Error escapes
                        *
                        ********************************************************************************
00000178 7008       G_SCBSY   MOVEQ  #SC_BUSY,D0          sc is busy
0000017A 600E                 BRA.S  ESC_ERR
0000017C 7015       CRD_DOWN  MOVEQ  #CRD_DWN,D0          CARD IS DOWN
0000017E 600A                 BRA.S  ESC_ERR
00000180 7014       G_SC_ERR  MOVEQ  #BAD_SCT,D0          bad set/clear/test
00000182 6006                 BRA.S  ESC_ERR
00000184 7007       GTERR_B   MOVEQ  #TFR_ERR,D0          bad transfer specification
00000186 6002                 BRA.S  ESC_ERR
00000188 700D       GTERR_D   MOVEQ  #NO_DMA,D0           DMA not installed
                        *             BRA.S  ESC_ERR


0000018A 48C0       ESC_ERR   EXT.L  D0                   assume errors<128
0000018C 2B40 FFBE            MOVE.L D0,IOE_RSLT(A5)      save ioe_result
00000190 102A 002D            MOVE.B IO_SC(A2),D0
00000194 2B40 FFBA            MOVE.L D0,IOE_SC(A5)        save io s.c.
00000198 3B7C FFE6            MOVE.W #IOE_ERROR,ESC_CODE(A5)
                FFFE
0000019E 4E4A                 TRAP   #10                  escape
```

```
                        ********************************************
                        *
                        *       LINE DEFINITIONS
                        *
                        *           bit 0       STI0
                        *               1       STI1
                        *               2       EIR
                        *               3       PSTS
                        *               4       CTL0
                        *               5       CTL1
                        *               6       READY
                        *               7       PCTL
                        *
                        ********************************************


                        ********************************************
                        *
                        *       SET LINE
                        *
                        ********************************************
000001A0 C23C 0007 G_SET    AND.B  #7,D1            MASK TO RIGHT SIZE
000001A4 B23C 0005          CMP.B  #5,D1            \ HANDLE SET PCTL
000001A8 6714               BEQ.S  GSET_1           /
000001AA B23C 0006          CMP.B  #6,D1            \ CAN ONLY SET
000001AE 6DD0               BLT.S  G_SC_ERR         /     CTL0/1
000001B0 923C 0005          SUB.B  #5,D1            GET INTO 1/2 FORM
000001B4 8329 0007          OR.B   D1,7(A1)         SET CTL0 or 1
000001B8 832A 002C          OR.B   D1,EIRB_OFF(A2) SAVE IN EIRB
000001BC 4E75               RTS
000001BE 1341 0000 GSET_1   MOVE.B D1,0(A1)         SET PCTL
000001C2 4E75               RTS


                        ********************************************
                        *
                        *       CLEAR LINE
                        *
                        ********************************************
000001C4 C23C 0007 G_CLR    AND.B  #7,D1            MASK TO RIGHT SIZE
000001C8 B23C 0006          CMP.B  #6,D1            \ CAN ONLY CLEAR
000001CC 6DB2               BLT.S  G_SC_ERR         /     CTL0/1
000001CE 102A 002C          MOVE.B EIRB_OFF(A2),D0 GET OLD CTL0/1
000001D2 923C 0006          SUB.B  #6,D1            GET INTO 0/1 FORM
000001D6 0380               BCLR   D1,D0            CLEAR COPY
000001D8 1540 002C          MOVE.B D0,EIRB_OFF(A2) SAVE IN EIRB
000001DC 1340 0007          MOVE.B D0,7(A1)         WRITE TO CARD
000001E0 4E75               RTS


                        ********************************************
                        *
                        *       TEST LINE
                        *
```



```
                        ********************************************
000001E2 4240       G_TEST  CLR.W   D0              set FALSE indication
000001E4 C23C 0007          AND.B  #7,D1            mask to the right size
000001E8 B23C 0007          CMP.B  #7,D1            \ CHECK FOR PCTL
000001EC 6792               BEQ    G_SC_ERR         /       AND GIVE ERROR
000001EE B23C 0003          CMP.B  #3,D1
000001F2 6E0A               BGT.S  GTST2
000001F4 0329 0007 GTST1    BTST   D1,7(A1)         TEST STO/1,EIR,or psts
000001F8 6702       GTST_CHK BEQ.S  GTST_RTS        if clear then RTS
000001FA 7001       GTST_SET MOVEQ  #1,D0               else return true indication
000001FC 4E75       GTST_RTS RTS


000001FE 923C 0004 GTST2    SUB.B  #4,D1
00000202 B23C 0001          CMP.B  #1,D1
00000206 6E06               BGT.S  GTST4
00000208 032A 002C GTST3    BTST   D1,EIRB_OFF(A2) TEST CTL0/1
0000020C 60EA               BRA.S  GTST_CHK         GO TO COMMON CHECK CODE


0000020E 0829 0000 GTST4    BTST   #0,0(A1)         CHECK READY LINE
         0000
00000214 60E2               BRA.S  GTST_CHK
```

```
                        ********************************************************************************
                        *
                        *       G_RDS
                        *
                        *         READ STATUS
                        *
                        *         PASCAL ROUTINE
                        *
                        ********************************************************************************
              0000 0002 G_ROUTINE EQU    2
              0000 0001 G_TEMP    EQU    1
              0000 0000 G_CRDREG  EQU    0
                        *
                        *
00000216 41FA 004E      G_RDS     LEA    G_RDSTBL,A0          get pointer to lookup table
0000021A D241                     ADD.W  D1,D1                multiply the rds register by 2
0000021C B23C 000C                CMP.B  #G_RT_SIZ,D1         \ check for out of bounds
00000220 6C18                     BGE.S  RDS_ERR              /
00000222 3030 1000                MOVE.W 0(A0,D1),D0          get the table entry
00000226 6B12                     BMI.S  RDS_ERR              if the entry is 0 then error
00000228 B03C 0001                CMP.B  #G_TEMP,D0
                        * tm      BEQ.S  GR_TEMP
0000022C 6D12                     BLT.S  GR_CARD
0000022E E048                     LSR    #8,D0                get the routine offset
00000230 671A                     BEQ.S  G_RDS_DI             -  status rtn 3 - data in
00000232 5340                     SUBQ   #1,D0
00000234 671C                     BEQ.S  G_RDS_RDY            -  status 4 - ready
00000236 5340                     SUBQ   #1,D0
00000238 6722                     BEQ.S  G_RDS_PST            -  status  5 - peripheral status
                        *         BRA.S  RDS_ERR              there are no more status 'routines'


0000023A 7013           RDS_ERR   MOVEQ  #BAD_RDS,D0          bad read status
0000023C 6000 FF4C                BRA    ESC_ERR


                        *
                        *         retrieve temps as words
                        *
                        * tm GR_TEMP   LSR    #8,D0                get temp offset
                        * tm           MOVE.W 0(A2,D0),D0          get the value
                        * tm           RTS


                        *
                        *         retrieve card registers as bytes
                        *
00000240 E048           GR_CARD   LSR    #8,D0                get the card offset
00000242 1031 0000      GR_CARD1  MOVE.B 0(A1,D0),D0          get the value
00000246 0240 00FF      GREGEXIT  ANDI.W #$00FF,D0            mask off garbage
0000024A 4E75                     RTS


                        *
                        *         data in
```



```
                        *
0000024C 3029 0004      G_RDS_DI  MOVE.W 4(A1),D0              get data lines
00000250 4E75                     RTS


00000252 1029 0000      G_RDS_RDY MOVE.B 0(A1),D0              get ready line
00000256 0200 0001                ANDI.B #$01,D0              mask to 1 bit
0000025A 60EA                     BRA    GREGEXIT             and get out


0000025C 1029 0007      G_RDS_PST MOVE.B 7(A1),D0              get status
00000260 0200 000F                ANDI.B #$0F,D0              mask to 4 bits
00000264 60E0                     BRA    GREGEXIT             and get out


              0000 0266 G_RDSTBL  EQU    *
00000266 01                       DC.B   1,G_CRDREG           status 0 - card reg  0   - card id
00000267 00
00000268 03                       DC.B   3,G_CRDREG           status 1 - card reg 3   - intr/dma status
00000269 00
0000026A 09                       DC.B   9,G_ROUTINE          status 2 - not implemented
0000026B 02
0000026C 00                       DC.B   0,G_ROUTINE          status 3 - data in
0000026D 02
0000026E 01                       DC.B   1,G_ROUTINE          status 4 - ready line
0000026F 02
00000270 02                       DC.B   2,G_ROUTINE          status 5 - peripheral status
00000271 02
              0000 0272 G_RT_END  EQU    *
              0000 000C G_RT_SIZ  EQU    G_RT_END-G_RDSTBL    size of table
```

```
                        ********************************************************************************
                        *
                        *       G_WTC
                        *
                        *         WRITE CONTROL
                        *
                        *         ENTRY:  D0.W = PARAMETER
                        *
                        ********************************************************************************
00000272 B27C 0006 G_WTC      CMP.W   #6,D1                   \
00000276 6CC2                 BGE.S   RDS_ERR                 ) check wtc limits
00000278 48C1                 EXT.L   D1
0000027A D281                 ADD.L   D1,D1
0000027C 4EFB 1002            JMP     GWTCTBL(D1)

00000280 600A       GWTCTBL   BRA.S   G_WTC_RST               CONTROL 0 - D0 A RESET
00000282 600C                 BRA.S   G_SET_PCT               CONTROL 1 - set pctl
00000284 6010                 BRA.S   G_SET_CTL               CONTROL 2 : set control lines
00000286 601C                 BRA.S   G_DATA_O                CONTROL 3 - write data out
00000288 60B0                 BRA.S   RDS_ERR                 CONTROL 4 : not used
0000028A 601E                 BRA.S   G_EIR                   CONTROL 5 : enable intrpts


0000028C 6000 FE54 G_WTC_RST BRA      G_INIT                  reset card

00000290 1340 0000 G_SET_PCT MOVE.B D0,0(A1)                  set pctl line           SPR836 TM 8/9/82
00000294 4E75                RTS

00000296 1340 0007 G_SET_CTL MOVE.B D0,7(A1)                  set ctl 0 and 1
0000029A C03C 0003           AND.B  #3,D0                     \ save CTL0/1           508 TM 9/17/82
0000029E 812A 002C           OR.B   D0,EIRB_OFF(A2)           ) in EIR byte           508 TM 9/17/82
000002A2 4E75                RTS
000002A4 3340 0004 G_DATA_O  MOVE.W D0,4(A1)                  write 16 bit data
000002A8 4E75                RTS
```



```
                        ************************************************
                        *
                        *  EIR
                        *
                        ************************************************
000002AA 1540 002C G_EIR      MOVE.B  D0,EIRB_OFF(A2)
000002AE 4EB9 0000            JSR     ITXFR                                           475 TM 9/17/82
              0000
000002B4 6616                 BNE.S   G_RTS                   if tfr then don't
000002B6 1340 0007 G_WTI5     MOVE.B  D0,7(A1)                Update CTL1:CTL0
000002BA 0800 0005            BTST    #5,D0                   Check RESET bit
000002BE 6600 FE22            BNE     G_RESET
000002C2 4A00                 TST.B   D0
000002C4 6C08                 BGE.S   G_RTS1
000002C6 137C 0080            MOVE.B  #$80,3(A1)
              0003
000002CC 4E75           G_RTS RTS
000002CE 137C 0000 G_RTS1     MOVE.B  #0,3(A1)
              0003
000002D4 4E75                 RTS
```

```
                        ************************************************
                        *
                        *  GPIO tfr
                        *
                        ************************************************
000002D6 4EB9 0000 G_TFR    JSR    CHECK_TFR              wait for tfr to finish  475 TM 9/17/82
                 0000
000002DC 6100 FE70          BSR    G_WAIT                 Wait for FLG            xxx TM 10/6/82
000002E0 4A2B 000B          TST.B  TEND_OFF(A3)           \ end NOT ALLOWED
000002E4 6600 FE9E          BNE    GTERR_B                /
000002E8 202B 0010          MOVE.L TCNT_OFF(A3),D0        GET COUNT
000002EC 4241               CLR.W  D1                     \
000002EE 122B 0009          MOVE.B TUSR_OFF(A3),D1         \   COMPUTE OFFSEET INTO JUMP TABLE
000002F2 D241               ADD.W  D1,D1                   /
000002F4 4EB9 0000          JSR    TESTDMA                 /  BASED ON TFR        475 TM 9/17/82
                 0000
000002FA 6704               BEQ.S  G_NODMA                /      TYPE AND DMA PRESENCE
000002FC 0641 0014          ADDI.W #20,D1                 \
00000300 41FA 0008 G_NODMA  LEA    G_TBL,A0               \
00000304 D0F0 1000          ADDA.W 0(A0,D1),A0             INDEXED JUMP THRU TABLE
00000308 4ED0               JMP    (A0)                   /
                        *
                        *       TRANSFER JUMP TABLE
                        *
                        *                                  --------------------  DMA is not installed or available
0000030A FE7A       G_TBL   DC.W   GTERR_B-G_TBL  serial  interrupt
0000030C FE7E               DC.W   GTERR_D-G_TBL  serial  dma
0000030E 00F4               DC.W   G_T_FHS-G_TBL  serial  fhs
00000310 00F4               DC.W   G_T_FHS-G_TBL  serial  fastest
00000312 FE7A               DC.W   GTERR_B-G_TBL  serial  overlap
                        *                         --------------------
00000314 00C0               DC.W   G_T_INT-G_TBL  overlap interrupt
00000316 FE7E               DC.W   GTERR_D-G_TBL  overlap dma
00000318 00C8               DC.W   G_T_BST-G_TBL  overlap fhs
0000031A 00C8               DC.W   G_T_BST-G_TBL  overlap fastest
0000031C 00C0               DC.W   G_T_INT-G_TBL  overlap overlap
                        *                         --------------------  DMA is installed
0000031E FE7A               DC.W   GTERR_B-G_TBL  serial  interrupt
00000320 0028               DC.W   G_T_DMA-G_TBL  serial  dma
00000322 00F4               DC.W   G_T_FHS-G_TBL  serial  fhs
00000324 0028               DC.W   G_T_DMA-G_TBL  serial  fastest
00000326 FE7A               DC.W   GTERR_B-G_TBL  serial  overlap
                        *                         --------------------
00000328 00C0               DC.W   G_T_INT-G_TBL  overlap interrupt
0000032A 0028               DC.W   G_T_DMA-G_TBL  overlap dma
0000032C 00C8               DC.W   G_T_BST-G_TBL  overlap fhs
0000032E 0028               DC.W   G_T_DMA-G_TBL  overlap fastest
00000330 0028               DC.W   G_T_DMA-G_TBL  overlap overlap
```



```
                        *
                        *       Transfer DMA
                        *
00000332 B0BC 0000 G_T_DMA  CMP.L  #1,D0                  \ USE INTR IF COUNT=1 ON DMA
                 0001
00000338 6700 0090          BEQ    G_T_INT                /
0000033C 177C 0002          MOVE.B #TT_DMA,TACT_OFF(A3)   set tfr type to DMA
                 0007
00000342 4A2B 000D          TST.B  TDIR_OFF(A3)           \ test for transfer direction
00000346 6E36               BGT.S  G_TOD                  /
                        *
                        *       Transfer Input Dma:
                        *
                 0000 0348 G_TID    EQU    *
00000348 5380               SUBQ.L #1,D0                  0 - Set up DMA for Input
0000034A 4EB9 0000          JSR    GETDMA                 Tfr N-1 bytes via DMA   475 TM 9/17/82
                 0000
00000350 0829 0003          BTST   #3,3(A1)               Is BURST bit set on GPIO card ?
                 0003
00000356 6704               BEQ.S  G_DMA1                 No
00000358 0042 0008          ORI.W  #8,D2                  Yes: Set BURST bit in DMA arm byte
0000035C 3882       G_DMA1  MOVE   D2,(A4)                Arm DMA
0000035E 6100 005E          BSR    GD_STBSY               Set buffer busy (Card is still not triggered)
00000362 4A69 0004          TST    4(A1)                  rdi 4
00000366 1280               MOVE.B D0,(A1)                wti 7,0  (trigger card)
00000368 137C 0001          MOVE.B #1,2(A1)               RDYEN = 0 (so transfers won't interrupt)
                 0002
0000036E 4A2B 000A          TST.B  T_BW_OFF(A3)           Byte (0) or Word (1) transfers ?
00000372 6704               BEQ.S  G_DMA3
00000374 D67C 0004          ADD    #4,D3
00000378 1343 0003 G_DMA3   MOVE.B D3,3(A1)               Tell the GPIO what channel he's got and
                        *                                    watch the shit hit the fan
0000037C 602C               BRA.S  G_DMA_W                DONE
                        *
                        *       Transfer Output Dma:
                        *
                 0000 037E G_TOD    EQU    *
0000037E 4EB9 0000 G_DMAOUT JSR    GETDMA                 Get a DMA channel       475 TM 9/17/82
                 0000
00000384 0829 0003          BTST   #3,3(A1)               Is Burst mode enabled (1)
                 0003
0000038A 6704               BEQ.S  G_DMA4
0000038C 0042 0008          ORI.W  #8,D2                  Yes: Set the Burst bit in DMA arm byte
00000390 3882       G_DMA4  MOVE   D2,(A4)                Arm the DMA channel
00000392 6100 002A          BSR    GD_STBSY               Set the buffer busy
00000396 137C 0001          MOVE.B #1,2(A1)               Disable the transfer interrupt mechanism
                 0002
0000039C 4A2B 000A          TST.B  T_BW_OFF(A3)           Byte (1) or Word (0) transfers ?
000003A0 6704               BEQ.S  G_DMA6                 Byte: Don't set the WORD bit on GPIO card
000003A2 0043 0004          ORI.W  #4,D3                  WORD: Set the WORD bit
000003A6 1343 0003 G_DMA6   MOVE.B D3,3(A1)               Tell the GPIO what channel he's got, etc.
                        *
                        *       G_DMA_W                   IF SERIAL THEN WAIT FOR  COMPLETION
                        *
000003AA 182B 0009 G_DMA_W  MOVE.B TUSR_OFF(A3),D4        \
000003AE B83C 0005          CMP.B  #5,D4                   IS THE TRANSFER OVERLAP ?
000003B2 6C08               BGE.S  G_DMA_W2               /
```

```
862    000003B4 0C2B 00FF G_DMA_W1  CMPI.B #255,T_SC_OFF(A3)      IF NOT THEN WAIT TILL DONE
               0005
863    000003BA 66F8                BNE.S  G_DMA_W1
864    000003BC 4E75       G_DMA_W2 RTS
865
865
865
865
865
866    000003BE 49FA FC5E GD_STBSY  LEA    EXTG_EG_TDMA,A4       \ save g_dmaterm routine
867    000003C2 4EB9 0000           JSR    DMA_STBSY             / in dma temps          475 TM 9/17/82
               0000
868    000003C8 4E75                RTS                                                  475 TM 9/17/82
```



```
870                          *
871                          *         Transfer INTERRUPT
872                          *
873    000003CA 177C 0001 G_T_INT   MOVE.B #TT_INT,TACT_OFF(A3)   set tfr type to INTERRUPT
               0007
874    000003D0 6006                BRA.S  G_T_BIC                go to common code
875
875
875
876                          *
877                          *         Transfer BURST ( intr on 1st byte FHS on rest )
878                          *
879    000003D2 177C 0003 G_T_BST   MOVE.B #TT_BURST,TACT_OFF(A3) set tfr type to BURST
               0007
880                          *         BRA.S  G_T_BIC                go to common code
881
881
881
882                          *
883                          *         common interrupt and burst code
884                          *
885    000003D8 4EB9 0000 G_T_BIC   JSR    STBSY                 SET BUFFER BUSY, ETC
               0000
886    000003DE 4A2B 000D           TST.B  TDIR_OFF(A3)          \ test for transfer direction
887    000003E2 6E06                BGT.S  G_TOI                 /
888                          *
889                          *         Transfer Input Interrupt or Transfer Input Burst
890                          *
891              0000 03E4 G_TII     EQU    *
892    000003E4 4A69 0004 G_INT_I   TST    4(A1)                 Dummy read to set I/O line to I
893    000003E8 1280                MOVE.B D0,(A1)               wti 7,0 (trigger input)
894                          *
895                          *         Transfer Output Interrupt or Transfer Output Burst
896                          *
897              0000 03EA G_TOI     EQU    *
898    000003EA 137C 0003 G_INT_O   MOVE.B #3,2(A1)              Allow I/O to cause an interrupt
               0002
899    000003F0 102A 002C           MOVE.B EIRB_OFF(A2),D0       GET CTL0/1
900    000003F4 0000 0080           ORI.B  #$80_D0
901    000003F8 6100 FEBC           BSR    G_WTI5                Enable interrupts
902    000003FC 60AC                BRA    G_DMA_W               and wait if necessary
```

```
                        *
                        *          Transfer FHS
                        *
000003FE 177C 0004 G_T_FHS  MOVE.B #TT_FHS,TACT_OFF(A3)    set tfr type to FHS
         0007
00000404 4EB9 0000          JSR    STBSY                  set buffer busy          475 TM 9/17/82
         0000
0000040A 4EB9 0000          JSR    ITXFR                  get all pointers         475 TM 9/17/82
         0000
00000410 4A2B 000D          TST.B  TDIR_OFF(A3)           \ test for transfer direction
00000414 6E54               BGT.S  G_TFO                  /
                        *
                        *          Transfer FHS in
                        *
00000416 4A69 0004 G_TFI    TST    4(A1)                  to set I/O line to I     507 TM 9/17/82
0000041A 1280               MOVE.B D0,(A1)                trigger input            507 TM 9/17/82
0000041C 7000               MOVEQ  #0,D0                  clear upper part  of data in
0000041E 206B 0020          MOVEA.L TFIL_OFF(A3),A0       GET FILL POINTER
00000422 4A2B 000A          TST.B  T_BW_OFF(A3)           GET B/W INDICATIONS
00000426 661C               BNE.S  G_TFIW
00000428 6100 FD24 G_TFIB   BSR    G_WAIT
0000042C 1029 0005 G_TFIB1  MOVE.B 5(A1),D0               D0.L = Byte received
00000430 10C0               MOVE.B D0,(A0)+
00000432 5383               SUBQ.L #1,D3                  Decrement transfer counter
00000434 6F24               BLE.S  GTFI_TRM               If buffer full; exit fast handshake
00000436 B440               CMP.W  D0,D2                  If input character matches end character
00000438 6720               BEQ.S  GTFI_TRM                 then we're done
0000043A 1280               MOVE.B D0,(A1)                wti 7,0 (Trigger next input)
0000043C 0811 0000 G_TFIB2  BTST   #0,(A1)                PFLG = 1 ?
00000440 66EA               BNE    G_TFIB1                Yes: Get next byte
00000442 60F8               BRA    G_TFIB2                No:  Keep checking


00000444 6100 FD08 G_TFIW   BSR    G_WAIT
00000448 30E9 0004 G_TFIW1  MOVE   4(A1),(A0)+            Copy word from GPIO to buffer
0000044C 5383               SUBQ.L #1,D3                  Decrement transfer counter
0000044E 6F0A               BLE.S  GTFI_TRM               If D3 = 0 we're done
00000450 1280               MOVE.B D0,(A1)                wti 7,0 (Trigger next input)
00000452 0811 0000 G_TFIW2  BTST   #0,(A1)                PFLG = 1
00000456 66F0               BNE    G_TFIW1
00000458 60F8               BRA    G_TFIW2                No:  Keep checking


                        *
                        * FHS TRANSFER TERMINATION
                        *
0000045A 2743 0010 GTFI_TRM MOVE.L  D3,TCNT_OFF(A3)       D3 has bytes not finished
0000045E 2748 0020          MOVE.L  A0,TFIL_OFF(A3)       update fill pointer
00000462 4EB9 0000 GTFO_TRM JSR     STCLR                 MARK BUFFER FINISHED     475 TM 9/17/82
         0000
00000468 4E75               RTS


                        *
```



```
                        *          Transfer FHS out
                        *
0000046A 206B 001C G_TFO    MOVEA.L TEMP_OFF(A3),A0       GET EMPTY POINTER
0000046E 4A2B 000A          TST.B   T_BW_OFF(A3)          GET B/W INDICATIONS
00000472 6616               BNE.S   G_TFOW
00000474 6100 FCD8 G_TFOB   BSR     G_WAIT                Wait for card ready
00000478 1358 0005 G_TFOB1  MOVE.B  (A0)+,5(A1)           Copy byte from buffer to GPIO
0000047C 1280               MOVE.B  D0,(A1)               wti 7,0 (Trigger next output)
0000047E 5383               SUBQ.L  #1,D3                 Decrement transfer count
00000480 6F1E               BLE.S   G_TFO3                If D3 = 0 then we're done
00000482 0811 0000 G_TFOB2  BTST    #0,(A1)               PFLG = 1 ?
00000486 66F0               BNE     G_TFOB1               Yes: Get next byte
00000488 60F8               BRA     G_TFOB2               No:  Keep checking


0000048A 6100 FCC2 G_TFOW   BSR     G_WAIT                wait for card ready
0000048E 3358 0004 G_TFOW1  MOVE.W  (A0)+,4(A1)           Copy word from buffer to GPIO
00000492 1280               MOVE.B  D0,(A1)               wti 7,0 (Trigger next output)
00000494 5383               SUBQ.L  #1,D3                 Decrement transfer count
00000496 6F08               BLE.S   G_TFO3                IF D3 = 0 then we're done
00000498 0811 0000 G_TFOW2  BTST    #0,(A1)               PFLG = 1 ?
0000049C 66F0               BNE     G_TFOW1               Yes: Get next byte
0000049E 60F8               BRA     G_TFOW2               No:  Keep checking


000004A0 2748 001C G_TFO3   MOVE.L  A0,TEMP_OFF(A3)       SAVE EMPTY PTR
000004A4 42AB 0010          CLR.L   TCNT_OFF(A3)          CLEAR COUNT
000004A8 60B8               BRA.S   GTFO_TRM
```

```
                        ****************************************************
                        *
                        *  UNDMA
                        *
                        *  Release the DMA channel associated with the GPIO
                        *  card when the transfer is done.
                        *
                        ****************************************************
             0000 04AA G_DMATERM       EQU *
000004AA 007C 2700 G_UNDMA  ORI       #$2700,SR           Disable all other interrupts
000004AE 4EB9 0000          JSR       DROPDMA             Release the DMA channel 475 TM 9/17/82
             0000
000004B4 4EB9 0000          JSR       ITXFR               See if buf was active   475 TM 9/17/82
             0000
000004BA 6756               BEQ.S     G_UNDEND            No: this should never happen
000004BC 4A2B 000D          TST.B     TDIR_OFF(A3)        Was it an Input (0) or Output (1) ?
000004C0 672A               BEQ.S     UNINPUT
000004C2 102A 002C UNOUTPUT MOVE.B    EIRB_OFF(A2),D0
000004C6 C07C 000F          AND       #$F,D0
000004CA 6100 FDEA          BSR       G_WTI5
000004CE 337C 0300          MOVE.W    #$300,2(A1)         Restore REDYN and EIREN
             0002
000004D4 4A2B 000A          TST.B     T_BW_OFF(A3)        Was it a Byte (1) or Word (0) tfr ?
000004D8 6702               BEQ.S     UNOUT2
000004DA D683               ADD.L     D3,D3               Word: Double the count
000004DC D7AB 001C UNOUT2   ADD.L     D3,TEMP_OFF(A3)     UPDATE EMPTY POINTER
000004E0 42AB 0010          CLR.L     TCNT_OFF(A3)        CLEAR COUNT
000004E4 4EB9 0000          JSR       STCLR               Unbusy the buffer       475 TM 9/17/82
             0000
000004EA 4E75               RTS
000004EC 202B 0010 UNINPUT  MOVE.L    TCNT_OFF(A3),D0     D0 = Transfer count
                        * tm MOVE.L    D4,TCNT_OFF(A3)     SET COUNT TO REMAINING BYTES
000004F0 9084               SUB.L     D4,D0               GET ACTUAL BYTES TFR'D
000004F2 4A2B 000A          TST.B     T_BW_OFF(A3)        Was it a Byte (1) or Word (0) tfr ?
000004F6 6702               BEQ.S     UNIN2
000004F8 D080               ADD.L     D0,D0               Word: Double the count
000004FA D1AB 0020 UNIN2    ADD.L     D0,TFIL_OFF(A3)     UPDATE FILL POINTER
000004FE 277C 0000          MOVE.L    #1,TCNT_OFF(A3)     Last byte is received via interrupt
             0001 0010
00000506 337C 0380          MOVE.W    #$380,2(A1)         Allow it to cause an interrupt
             0002
0000050C 177C 0001          MOVE.B    #TT_INT,TACT_OFF(A3)  change tfr type
             0007
00000512 4E75       G_UNDEND RTS
```



```
                        ****************************************************
                        *
                        *  GPIO INTERRUPT SERVICE ROUTINE
                        *
                        ****************************************************
             0000 0514 G_ISR     EQU     *
00000514 4EB9 0000 G_ISR1   JSR     ITXFR               Transfer in progress?   475 TM 9/17/82
             0000
0000051A 6614               BNE.S   G_ATFR
0000051C 102A 002C G_ISR2   MOVE.B  EIRB_OFF(A2),D0     NO TFR IN PROGRESS
00000520 C07C 000F          AND     #$F,D0
00000524 6100 FD90          BSR     G_WTI5
00000528 4EB9 0000          JSR     LOGINT              log the interrupt       475 TM 9/17/82
             0000
0000052E 4E75       GISR_END RTS
00000530 1029 0003 G_ATFR   MOVE.B  3(A1),D0            Put DMA channel for GPIO in D0
00000534 C07C 0003          AND     #3,D0
00000538 6648               BNE.S   G_TABORT            If DMA in progress: Abort & update pointers
0000053A B23C 0004          CMP.B   #TT_FHS,D1          \  If FHS then exit
0000053E 6742               BEQ.S   G_TABORT            /
00000540 923C 0001 G_ATFR1  SUB.B   #1,D1               \ IF INTR THEN D1=0
00000544 E509               LSL.B   #2,D1               \    BURST THEN D1=8
00000546 4A2B 000A          TST.B   T_BW_OFF(A3)        \ IF WORD THEN ADD 4
0000054A 6704               BEQ.S   GTST_DIR            /
0000054C D23C 0004          ADD.B   #4,D1
00000550 4A2B 000D GTST_DIR TST.B   TDIR_OFF(A3)        \ IF OUTPUT THEN ADD 16
00000554 6708               BEQ.S   G_ATFR2             /
00000556 D23C 0010          ADD.B   #$10,D1
0000055A 4881               EXT.W   D1
0000055C 48C1               EXT.L   D1
0000055E 4EFB 1002 G_ATFR2  JMP     G_TTBL(D1)          computed goto
00000562 6000 00C0 G_TTBL   BRA     G_TIIB              INTR IN BYTE
00000566 6000 00D0          BRA     G_TIIW              INTR IN WORD
0000056A 6000 0084          BRA     G_TIFB              BRST IN BYTE
0000056E 6000 009E          BRA     G_TIFW              BRST IN WORD
00000572 6000 0044          BRA     G_TOIB              INTR OUT BYTE
00000576 6000 0046          BRA     G_TOIW              INTR OUT WORD
0000057A 6000 0010          BRA     G_TOFB              BRST OUT BYTE
0000057E 6000 0022          BRA     G_TOFW              BRST OUT WORD
00000582 4EB9 0000 G_TABORT JSR     DROPDMA             Release DMA channel     475 TM 9/17/82
             0000
00000588 6000 00C2          BRA     G_TDIN
```

```
1062                        ***********************************************
1063                        *
1064                        *       OUTPUT TRANSFERS
1065                        *
1066                        ***********************************************
1067    0000058C 007C 2700 G_TOFB   ORI     #$2700,SR               Disable interrupts
1068    00000590 1358 0005 G_TOFB1  MOVE.B  (A0)+,5(A1)             Copy byte from buffer to GPIO
1069    00000594 5383               SUBQ.L  #1,D3                   Decrement transfer count
1070    00000596 6F3A               BLE.S   G_TDOUT                 If D3 = 0 then we're done
1071    00000598 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger next output)
1072    0000059A 0811 0000 G_TOFB2  BTST    #0,(A1)                 PFLG = 1 ?
1073    0000059E 66F0               BNE     G_TOFB1                 Yes: Get next byte
1074    000005A0 60F8               BRA     G_TOFB2                 No:  Keep checking
1075
1075
1075
1075
1076    000005A2 007C 2700 G_TOFW   ORI     #$2700,SR               Disable interrupts
1077    000005A6 3358 0004 G_TOFW1  MOVE.W  (A0)+,4(A1)             Copy word from buffer to GPIO
1078    000005AA 5383               SUBQ.L  #1,D3                   Decrement transfer count
1079    000005AC 6F24               BLE.S   G_TDOUT                 IF D3 = 0 then we're done
1080    000005AE 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger next output)
1081    000005B0 0811 0000 G_TOFW2  BTST    #0,(A1)                 PFLG = 1 ?
1082    000005B4 66F0               BNE     G_TOFW1                 Yes: Get next byte
1083    000005B6 60F8               BRA     G_TOFW2                 No:  Keep checking
1084
1084
1084
1084
1085    000005B8 1358 0005 G_TOIB   MOVE.B  (A0)+,5(A1)             Copy a byte from buffer to GPIO
1086    000005BC 6004               BRA.S   G_ENDOUT
1087    000005BE 3358 0004 G_TOIW   MOVE.W  (A0)+,4(A1)             Copy a word from buffer to GPIO
1088    000005C2 2748 001C G_ENDOUT MOVE.L  A0,TEMP_OFF(A3)         Save A0 for use on next output byte/word
1089    000005C6 5383               SUBQ.L  #1,D3
1090    000005C8 2743 0010          MOVE.L  D3,TCNT_OFF(A3)         Decrement transfer out counter
1091    000005CC 6F04               BLE.S   G_TDOUT                 If zero, go thru Transfer Done Out
1092    000005CE 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger byte out)
1093    000005D0 4E75               RTS                             End of ISR
1094
1094
1094
1094
1095    000005D2 2748 001C G_TDOUT  MOVE.L  A0,TEMP_OFF(A3)         save empty ptr
1096    000005D6 42AB 0010          CLR.L   TCNT_OFF(A3)            clear tfr count
1097    000005DA 137C 0000          MOVE.B  #0,3(A1)                Stop card from interrupting
                 0003
1098    000005E0 136A 002C          MOVE.B  EIRB_OFF(A2),7(A1)      Put EIR byte in CTL1/CTL0
                 0007
1099    000005E6 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger last output)
1100    000005E8 4EB9 0000          JSR     STCLR                   Unbusy buffer           475 TM 9/17/82
                 0000
1101    000005EE 4E75               RTS
```



```
1103                        ***********************************************
1104                        *
1105                        *       INPUT TRANSFERS
1106                        *
1107                        ***********************************************
1108    000005F0 007C 2700 G_TIFB   ORI     #$2700,SR
1109    000005F4 7000               MOVEQ   #0,D0
1110    000005F6 1029 0005 G_TIFB1  MOVE.B  5(A1),D0                D0.L = Byte received
1111    000005FA 10C0               MOVE.B  D0,(A0)+
1112    000005FC 5383               SUBQ.L  #1,D3                   Decrement transfer counter
1113    000005FE 6F4C               BLE.S   G_TDIN                  If buffer full; exit fast handshake
1114    00000600 B440               CMP.W   D0,D2                   If input character matches end character
1115    00000602 6748               BEQ.S   G_TDIN                    then we're done
1116    00000604 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger next input)
1117    00000606 0811 0000 G_TIFB2  BTST    #0,(A1)                 PFLG = 1 ?
1118    0000060A 66EA               BNE     G_TIFB1                 Yes: Get next byte
1119    0000060C 60F8               BRA     G_TIFB2                 No:  Keep checking
1120
1120
1120
1120
1121    0000060E 007C 2700 G_TIFW   ORI     #$2700,SR               Disable interrupts
1122    00000612 30E9 0004 G_TIFW1  MOVE    4(A1),(A0)+             Copy word from GPIO to buffer
1123    00000616 5383               SUBQ.L  #1,D3                   Decrement transfer counter
1124    00000618 6F32               BLE.S   G_TDIN                  If D3 = 0 we're done
1125    0000061A 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger next input)
1126    0000061C 0811 0000 G_TIFW2  BTST    #0,(A1)                 PFLG = 1
1127    00000620 66F0               BNE     G_TIFW1
1128    00000622 60F8               BRA     G_TIFW2                 No:  Keep checking
1129
1129
1129
1129
1130    00000624 7000       G_TIIB  MOVEQ   #0,D0
1131    00000626 1029 0005          MOVE.B  5(A1),D0                D0.L = Byte received
1132    0000062A 10C0               MOVE.B  D0,(A0)+                Store byte in input buffer
1133    0000062C B440               CMP.W   D0,D2                   Compare termination byte with input byte
1134    0000062E 660C               BNE.S   G_ENDIN                 No match: Everybody get out of here !
1135    00000630 5383               SUBQ.L  #1,D3                   Decrement byte counter
1136    00000632 2743 0010          MOVE.L  D3,TCNT_OFF(A3)
1137    00000636 6014               BRA.S   G_TDIN
1138
1138
1138
1138
1139    00000638 30E9 0004 G_TIIW   MOVE    4(A1),(A0)+             Copy word from GPIO to buffer
1140    0000063C 2748 0020 G_ENDIN  MOVE.L  A0,TFIL_OFF(A3)         Save buffer pointer
1141    00000640 5383               SUBQ.L  #1,D3
1142    00000642 2743 0010          MOVE.L  D3,TCNT_OFF(A3)
1143    00000646 6F04               BLE.S   G_TDIN
1144    00000648 1280               MOVE.B  D0,(A1)                 wti 7,0
1145    0000064A 4E75               RTS
1146
1146
1146
1146
1147    0000064C 2748 0020 G_TDIN   MOVE.L  A0,TFIL_OFF(A3)         save fill ptr
```

```
1148    00000650 2743 0010          MOVE.L  D3,TCNT_OFF(A3)         save remaining count
1149    00000654 137C 0000          MOVE.B  #0,3(A1)                Stop card from interrupting
                 0003
1150    0000065A 136A 002C          MOVE.B  EIRB_OFF(A2),7(A1)      Put EIR byte in CTL1/CTL0
                 0007
1151    00000660 4EB9 0000          JSR     STCLR                   Unbusy buffer           475 TM 9/17/82
                 0000
1152    00000666 4E75               RTS
```



```
1154                                END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ABORT_IO	ABS	106	00000008
CHECK_TFR	ABS	113	00000019
DMA_STBSY	ABS	111	00000013
DROPDMA	ABS	109	0000000F
GETDMA	ABS	108	0000000D
IODECLARATIONS	ABS	264	0000001C
ITXFR	ABS	105	00000006
LOGINT	ABS	107	00000008
STBSY	ABS	103	00000002
STCLR	ABS	104	00000004
SYSGLOBALS	ABS	265	00000020
TESTDMA	ABS	110	00000011
WAIT_TFR	ABS	112	00000016

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM		VALUE
A0	AREG	0			00000000
A1	AREG	0			00000001
A2	AREG	0			00000002
A3	AREG	0			00000003
A4	AREG	0			00000004
A5	AREG	0			00000005
A6	AREG	0			00000006
A7	AREG	0			00000007
AVAIL_OFF	ABS	204			00000034
BADTMO	ABS	282			0000000B
BAD_RDS	ABS	290			00000013
BAD_SCT	ABS	291			00000014
BUFI_OFF	ABS	195			00000024
BUFO_OFF	ABS	196			00000028
BUF_BUSY	ABS	280			00000009
CCR	STREG	0			00000005
CRD_DOWN	REL	551			0000017C
CRD_DWN	ABS	292			00000015
C_ADR	ABS	194			00000020
D0	DREG	0			00000000
D1	DREG	0			00000001
D2	DREG	0			00000002
D3	DREG	0			00000003
D4	DREG	0			00000004
D5	DREG	0			00000005
D6	DREG	0			00000006
D7	DREG	0			00000007
EIRB_OFF	ABS	197			0000002C
EOD_SEEN	ABS	293			00000016
ESC_CODE	ABS	301	SYSGLOBALS	+	FFFFFFFE
ESC_ERR	REL	560			0000018A
EXTG_EG_CLR	REL	434			000000BC
EXTG_EG_INIT	REL	319			00000002
EXTG_EG_ISR	REL	328			00000010

EXTG_EG_RDB	REL	346	0000002C
EXTG_EG_RDS	REL	390	00000074
EXTG_EG_RDW	REL	368	00000050
EXTG_EG_SET	REL	424	000000AC
EXTG_EG_TDMA	REL	337	0000001E
EXTG_EG_TEST	REL	444	000000CC
EXTG_EG_TFR	REL	414	0000009C
EXTG_EG_WTB	REL	358	00000040
EXTG_EG_WTC	REL	403	0000008A
EXTG_EG_WTW	REL	380	00000064
EXTG_EXTG	REL	314	00000000
GD_STBSY	REL	866	000003BE
GISR_END	REL	1031	0000052E
GREGEXIT	REL	686	00000246
GR_CARD	REL	684	00000240
GR_CARD1	REL	685	00000242
GSET_1	REL	597	000001BE
GTERR_B	REL	555	00000184
GTERR_D	REL	557	00000188
GTFI_TRM	REL	945	0000045A
GTFO_TRM	REL	947	00000462
GTSTI	REL	626	000001F4
GTST2	REL	631	000001FE
GTST3	REL	634	00000208
GTST4	REL	637	0000020E
GTST_CHK	REL	627	000001F8
GTST_DIR	REL	1043	00000550
GTST_RTS	REL	629	000001FC
GTST_SET	REL	628	000001FA
GWTCTBL	REL	728	00000280
G_ATFR	REL	1033	00000530
G_ATFR1	REL	1038	00000540
G_ATFR2	REL	1048	0000055E
G_CLR	REL	605	000001C4
G_CRDREG	ABS	651	00000000
G_DATA_0	REL	744	000002A4
G_DMA1	REL	830	0000035C
G_DMA3	REL	838	00000378
G_DMA4	REL	849	00000390
G_DMA6	REL	855	000003A6
G_DMAOUT	REL	845	0000037E
G_DMATERM	REL	987	000004AA
G_DMA_W	REL	859	000003AA
G_DMA_W1	REL	862	000003B4
G_DMA_W2	REL	864	000003BC
G_EIR	REL	752	000002AA
G_ENDIN	REL	1140	0000063C
G_ENDOUT	REL	1088	000005C2
G_INIT	REL	461	000000E2
G_INT_I	REL	892	000003E4
G_INT_O	REL	898	000003EA
G_IOFS	REL	475	00000100
G_IOFS1	REL	479	0000010C
G_IOFS2	REL	482	00000114
G_ISR	REL	1024	00000514
G_ISR1	REL	1025	00000514
G_ISR2	REL	1027	0000051C

```
G_NODMA          REL      782                          00000300
G_RDB            REL      500                          00000122
G_RDS            REL      654                          00000216
G_RDSTBL         REL      703                          00000266
G_RDS_DI         REL      692                          0000024C
G_RDS_PST        REL      699                          0000025C
G_RDS_RDY        REL      695                          00000252
G_RESET          REL      462                          000000E2
G_ROUTINE        ABS      649                          00000002
G_RTS            REL      761                          000002CC
G_RTS1           REL      762                          000002CE
G_RT_END         REL      710                          00000272
G_RT_SIZ         ABS      711                          0000000C
G_SCBSY          REL      549                          00000178
G_SC_ERR         REL      553                          00000180
G_SET            REL      588                          000001A0
G_SET_CTL        REL      740                          00000296
G_SET_PCT        REL      737                          00000290
G_STS0           REL      510                          0000013A
G_STS1           REL      514                          00000146
G_STSCHK         REL      508                          00000134
G_TABORT         REL     1059                          00000582
G_TBL            REL      789                          0000030A
G_TDIN           REL     1147                          0000064C
G_TDOUT          REL     1095                          000005D2
G_TEMP           ABS      650                          00000001
G_TEST           REL      620                          000001E2
G_TFI            REL      915                          00000416
G_TFIB           REL      921                          00000428
G_TFIB1          REL      922                          0000042C
G_TFIB2          REL      929                          0000043C
G_TFIW           REL      933                          00000444
G_TFIW1          REL      934                          00000448
G_TFIW2          REL      938                          00000452
G_TFO            REL      953                          0000046A
G_TFO3           REL      974                          000004A0
G_TFOB           REL      956                          00000474
G_TFOB1          REL      957                          00000478
G_TFOB2          REL      961                          00000482
G_TFOW           REL      965                          0000048A
G_TFOW1          REL      966                          0000048E
G_TFOW2          REL      970                          00000498
G_TFR            REL      771                          000002D6
G_TID            REL      824                          00000348
G_TIFB           REL     1108                          000005F0
G_TIFB1          REL     1110                          000005F6
G_TIFB2          REL     1117                          00000606
G_TIFW           REL     1121                          0000060E
G_TIFW1          REL     1122                          00000612
G_TIFW2          REL     1126                          0000061C
G_TII            REL      891                          000003E4
G_TIIB           REL     1130                          00000624
G_TIIW           REL     1139                          00000638
G_TOD            REL      844                          0000037E
G_TOFB           REL     1067                          0000058C
G_TOFB1          REL     1068                          00000590
G_TOFB2          REL     1072                          0000059A
```



```
G_TOFW           REL     1076                          000005A2
G_TOFW1          REL     1077                          000005A6
G_TOFW2          REL     1081                          000005B0
G_TOI            REL      897                          000003EA
G_TOIB           REL     1085                          000005B8
G_TOIW           REL     1087                          000005BE
G_TTBL           REL     1050                          00000562
G_T_BIC          REL      885                          000003D8
G_T_BST          REL      879                          000003D2
G_T_DMA          REL      816                          00000332
G_T_FHS          REL      907                          000003FE
G_T_INT          REL      873                          000003CA
G_UNDEND         REL     1016                          00000512
G_UNDMA          REL      988                          000004AA
G_WAIT           REL      522                          0000014E
G_WAIT2          REL      526                          0000015A
G_WAIT3          REL      532                          00000162
G_WAIT4          REL      539                          00000170
G_WRTS           REL      541                          00000176
G_WTB            REL      489                          00000116
G_WTC            REL      722                          00000272
G_WTC_RST        REL      735                          0000028C
G_WTI5           REL      755                          000002B6
H_ISR_PM         ABS      193                          0000001C
H_ISR_PR         ABS      191                          00000014
H_ISR_SL         ABS      192                          00000018
IOE_ERROR        ABS      296                          FFFFFFE6
IOE_RSLT         ABS      298 IODECLARATIONS  +        FFFFFFBE
IOE_SC           ABS      299 IODECLARATIONS  +        FFFFFFBA
IO_MISC          ABS      294                          00000017
IO_SC            ABS      198                          0000002D
ISR_ENTRY        ABS      189                          00000000
MA               ABS      203                          00000033
MA_W             ABS      202                          00000032
NOT_HPIB         ABS      273                          00000002
NOT_LSTN         ABS      287                          00000010
NOT_TALK         ABS      286                          0000000F
NO_ACTL          ABS      274                          00000003
NO_CARD          ABS      272                          00000001
NO_DATA          ABS      277                          00000006
NO_DMA           ABS      284                          0000000D
NO_DRV           ABS      283                          0000000C
NO_DVC           ABS      275                          00000004
NO_SCTL          ABS      289                          00000012
NO_SPACE         ABS      276                          00000005
NO_WORD          ABS      285                          0000000E
RCVR_BLK         ABS      302 SYSGLOBALS      +        FFFFFFF6
RDS_ERR          REL      671                          0000023A
SC_BUSY          ABS      279                          00000008
SP               AREG       0                          00000007
SR               STREG      0                          00000006
TACT_OFF         ABS      219                          00000007
TBSZ_OFF         ABS      242                          00000018
TBUF_OFF         ABS      241                          00000014
TCHR_OFF         ABS      238                          0000000E
TCNTERR          ABS      281                          0000000A
TCNT_OFF         ABS      240                          00000010
```

```
TDIR_OFF        ABS      236                    0000000D
TEMP_OFF        ABS      243                    0000001C
TEND_OFF        ABS      234                    0000000B
TFIL_OFF        ABS      244                    00000020
TFR_ERR         ABS      278                    00000007
TIMEOUT         ABS      199                    0000002E
TMO_ERR         ABS      288                    00000011
TTMP_OFF        ABS      217                    00000000
TT_BURST        ABS      255                    00000003
TT_DMA          ABS      254                    00000002
TT_FHS          ABS      256                    00000004
TT_INT          ABS      253                    00000001
TUSR_OFF        ABS      220                    00000009
T_BW_OFF        ABS      232                    0000000A
T_DMAPRI        ABS      249                    00000030
T_PM_OFF        ABS      248                    0000002C
T_PR_OFF        ABS      245                    00000024
T_SC_OFF        ABS      218                    00000005
T_SL_OFF        ABS      247                    00000028
UNIN2           REL     1012                    000004FA
UNINPUT         REL     1006                    000004EC
UNOUT2          REL     1001                    000004DC
UNOUTPUT        REL      994                    000004C2
USER_ISR        ABS      190                    00000014
USP             STREG      0                    00000007
```

GPIODVR

Purpose

GPIODVR is the assembly language support routines for the 9885 flexible disc drive transfer method: F9885.

```
PASS 1 COMPLETE. ERRORS: 0
    1                                       nosyms
    2                                       mname   gp
    3
    4                           src     module gp;
    5                           src
    6                           src     import
    7                           src       sysglobals, mini;
    8                           src
    9                           src     export
   10                           src       type
   11                           src         gpiotype = { gpio interface card definition }
   12                           src           packed record case integer of
   13                           src             0: (direct byte access)
   14                           src               ( r0,r1,r2,r3,r4,r5,r6,r7: byte );
   15                           src             1: (read access)
   16                           src               ( (r0)  R0pad:0..127; ready:boolean;
   17                           src                 (r1)  R1pad:0..7; cardid:0..31;
   18                           src                 (r2)  R2pad: byte;
   19                           src                 (r3)  Renab,req:boolean; intlevel:0..3; burst,Rword,Rdmac1,Rdmac0: boolean;
   20                           src                 (r4)  Rdata:
   21                           src                 (r5)    shortint;
   22                           src                 (r6)  R6pad:byte;
   23                           src                 (r7)  R7pad:0..15; psts,eir,sti1,sti0:boolean );
   24                           src             2: (write access)
   25                           src               ( (r0)  setpct1:byte;
   26                           src                 (r1)  reset:byte;
   27                           src                 (r2)  W2pad:0..63; rdyen,eiren:boolean;
   28                           src                 (r3)  Wenab:boolean; W3pad:0..15; Wword,Wdmac1,Wdmac0: boolean;
   29                           src                 (r4)  Wdata:
   30                           src                 (r5)    shortint;
   31                           src                 (r6)  W6pad:byte;
   32                           src                 (r7)  W7pad:0..63; ctl1,ctl0:boolean )
   33                           src             end; { gpio interface card definition }
   34                           src
   35                           src           gpio_r3_type = (separate declaration for use with structured constants)
   36                           src             packed record
   37                           src               (r3)  Wenab:boolean; W3pad:0..15; Wword,Wdmac1,Wdmac0: boolean;
   38                           src             end;
   39                           src
   40                           src           dmachanneltype = packed array[0..7] of byte;
   41                           src
   42                           src         var
   43                           src           dma_port[5242880]: array[0..1] of dmachanneltype;
   44                           src
   45                           src         procedure gpioclear    (var gpio: gpiotype);
   46                           src         procedure gpiowordout  (var gpio: gpiotype; datum: shortint);
   47                           src         function  gpiowordin   (var gpio: gpiotype): shortint;
   48                           src         procedure gpiodmaout   (var gpio: gpiotype;
   49                           src                                 command: shortint;
   50                           src                                 enable_byte: gpio_r3_type;
   51                           src                                 var dma_channel: dmachanneltype;
   52                           src                                 buffer: charptr; length: integer);
   53                           src         procedure gpiodmain    (var gpio: gpiotype;
   54                           src                                 command: shortint;
   55                           src                                 enable_byte: gpio_r3_type;
   56                           src                                 var dma_channel: dmachanneltype;
   57                           src                                 buffer: charptr; length: integer);
   58                           src     end; {gpio}
```



```
   60                           *
   61                           * dmaout/dmain stack frame definitions
   62                           *
   63              0000 0000 olda6     equ     +0      (long)          old stack frame pointer
   64              0000 0004 radd      equ     +4      (long)          return address
   65              0000 0008 len       equ     +8      (long)          length of transfer in words
   66              0000 000C buf       equ     +12     (long)          address of buffer
   67              0000 0010 chan      equ     +16     (long)          dma channel base address
   68              0000 0014 enab      equ     +20     (byte)          gpio dma enable byte
   69              0000 0015 stackpad  equ     +21     (byte)          unused - caused by pushing byte on stack
   70              0000 0016 cmnd      equ     +22     (word)          disc command (read/write/verify)
   71              0000 0018 gpio      equ     +24     (long)          gpio card base address
   72
   73
   74                           *
   75                           * Def's & Ref's
   76                           *
   77                                   def     gp_gp
   78                                   def     gp_gpioclear
   79                                   def     gp_gpiowordout
   80                                   def     gp_gpiowordin
   81                                   def     gp_gpiodmaout
   82                                   def     gp_gpiodmain
   83
   84                                   refa    sysglobals
   85                                   refa    mini_ioresc
   86
   87                                   lmode   mini_ioresc
   88
   89
   90                           *
   91                           * module initialization routine
   92                           *
   93    00000000 4E75          gp_gp           rts
   94
   95
   96                           *
   97                           * ioresult assignments
   98                           *
   99              0000 0004 ztimeout      equ     4
  100              0000 0015 zcatchall     equ     21
  101
  102
  103                           *
  104                           * error exits
  105                           *
  106    00000002 7015          ioresc_catchall moveq   #zcatchall,d0           zcatchall ioresult
  107
  108    00000004 3F00          ioresc          move    d0,-(sp)                push the ioresult
  109    00000006 4EB9 0000                     jsr     mini_ioresc             set ioresult then escape(-10)
                  0000
  110
  111
  112    0000000C 3B7C FFF4 bus_error        move    #-12,sysglobals-2(a5)   set the escapecode
                  FFFE
  113    00000012 4E4A                       trap    #10                     escape
```

```
                        *
                        * gpiowaitready with 2 second timeout
                        *
00000014 203C 0003 2569 waitready       move.l  #206185,d0              timeout counter

0000001A 0829 0003 0007 waitready_loop  btst    #3,7(a1)                peripheral status?
00000020 66E0                           bne     ioresc_catchall         ioresc(zcatchall) if so
00000022 0811 0000                      btst    #0,(a1)                 ready?
00000026 6608                           bne.s   waitready_rts           branch if so
00000028 5380                           subq.l  #1,d0                   decrement the timeout counter
0000002A 6EEE                           bgt     waitready_loop          loop until timeout count expired

0000002C 7004                           moveq   #ztimeout,d0            ztimeout ioresult
0000002E 60D4                           bra     ioresc                  escape

00000030 4E75           waitready_rts   rts                             return


                        ********************************************************************************
                        *                                       gpioclear                              *
                        ********************************************************************************

00000032 205F           gp_gpioclear    movea.l (sp)+,a0                pop the return address
00000034 225F                           movea.l (sp)+,a1                pop the gpio card base address
00000036 1340 0001                      move.b  d0,1(a1)                reset the card
0000003A 4229 0007                      clr.b   7(a1)                   clear ctl1 & clt0
0000003E 7064                           moveq   #100,d0                 prepare to...
00000040 51C8 FFFE                      dbra    d0,*                    wait a while...
00000044 61CE                           bsr     waitready               before testing psts & ready
00000046 4ED0                           jmp     (a0)                    return


                        ********************************************************************************
                        *                                     gpiowordout                              *
                        ********************************************************************************

00000048 205F           gp_gpiowordout  movea.l (sp)+,a0                pop the return address
0000004A 226F 0002                      movea.l 2(sp),a1                gpio card base address
0000004E 61C4                           bsr     waitready               wait until ready
00000050 3357 0004                      move    (sp),4(a1)              output the datum
00000054 1280                           move.b  d0,(a1)                 set pctl
00000056 5C8F                           addq.l  #6,sp                   pop off the parameters
00000058 4ED0                           jmp     (a0)                    return


                        ********************************************************************************
                        *                                     gpiowordin                               *
                        ********************************************************************************

0000005A 205F           gp_gpiowordin   movea.l (sp)+,a0                pop the return address
0000005C 225F                           movea.l (sp)+,a1                gpio card base address
0000005E 61B4                           bsr     waitready               wait until ready
00000060 3EA9 0004                      move    4(a1),(sp)              input the datum
00000064 1280                           move.b  d0,(a1)                 set pctl (same manner as 98032 autohandshake)
00000066 4ED0                           jmp     (a0)                    return
```



```
                        ********************************************************************************
                        *                                     gpiodmaout                               *
                        ********************************************************************************

00000068 4E56 0000 gp_gpiodmaout   link    a6,#0                        create our own stack frame

0000006C 226E 0018                 movea.l gpio(a6),a1                  gpio card base address
00000070 61A2                      bsr     waitready                    wait for previous handshake to complete

00000072 4E4B                      trap    #11                          move into supervisor mode       (scs)
                        * scs      move    sr,-(sp)                     prepare to disable interrupts
00000074 007C 2700                 ori     #$2700,sr                    disable interrupts ********************

00000078 336E 0016 0004            move    cmnd(a6),4(a1)               disc command
0000007E 1280                      move.b  d0,(a1)                      set pctl
00000080 136E 0014 0003            move.b  enab(a6),3(a1)               enable the gpio card for dma

00000086 246E 0010                 movea.l chan(a6),a2                  dma channel base address
0000008A 24AE 000C                 move.l  buf(a6),(a2)                 set the dma address
0000008E 202E 0008                 move.l  len(a6),d0                   transfer length
00000092 5380                      subq.l  #1,d0                        length-1
00000094 3540 0004                 move    d0,4(a2)                     set the dma count
00000098 357C 0006 0006            move    #$0006,6(a2)                 arm the dma channel

0000009E 46DF                      move    (sp)+,sr                     re-enable interrupts ******************

000000A0 47E9 0007                 lea     7(a1),a3                     gpio register 7 address
000000A4 7003                      moveq   #3,d0                        psts bit (gpio register 7)
000000A6 49EA 0007                 lea     7(a2),a4                     dma status lower byte address
000000AA 7200                      moveq   #0,d1                        armed bit (dma status register)

000000AC 0113      do_loop         btst    d0,(a3)                      psts?
000000AE 6600 FF52                 bne     ioresc_catchall              ioresc(zcatchall) if so
000000B2 0314                      btst    d1,(a4)                      dma channel still armed?
000000B4 66F6                      bne     do_loop                      keep looping if so

000000B6 0C6A FFFF 0004            cmpi    #-1,4(a2)                    dma transfer complete normally?
000000BC 6600 FF4E                 bne     bus_error                    branch if not (bus error)

000000C0 6100 FF52                 bsr     waitready                    wait for the final handshake to complete

000000C4 4E5E                      unlk    a6                           remove our stack frame
000000C6 205F                      movea.l (sp)+,a0                     pop the return address
000000C8 DEFC 0014                 adda    #20,sp                       pop off the parameters
000000CC 4ED0                      jmp     (a0)                         return
```

```
 218                    ********************************************************************************
 219                    *                                  gpiodmain                                   *
 220                    ********************************************************************************
 221
 222  000000CE 4E56 0000 gp_gpiodmain    link    a6,#0                   create our own stack frame
 223
 224  000000D2 226E 0018                 movea.l gpio(a6),a1             gpio card base address
 225  000000D6 6100 FF3C                 bsr     waitready               wait for previous handshake to complete
 226
 227  000000DA 4E4B                      trap    #11                     move into supervisor mode       (scs)
 228                         * scs       move    sr,-(sp)                prepare to disable interrupts
 229  000000DC 007C 2700                 ori     #$2700,sr               disable interrupts ********************
 230
 231  000000E0 336E 0016                 move    cmnd(a6),4(a1)          disc command
               0004
 232  000000E6 1280                      move.b  d0,(a1)                 set pctl
 233
 234  000000E8 7000                      moveq   #0,d0                   ready bit (register 0)
 235  000000EA 45E9 0007                 lea     7(a1),a2                register 7 address
 236  000000EE 7203                      moveq   #3,d1                   psts bit (register 7)
 237
 238  000000F0 0312      d_loop          btst    d1,(a2)                 peripheral status?
 239  000000F2 660A                      bne.s   d_enab                  fall out of the critical section if so
 240  000000F4 0111                      btst    d0,(a1)                 ready?
 241  000000F6 67F8                      beq     d_loop                  branch if not
 242
 243  000000F8 4269 0004                 clr     4(a1)                   clear the output buffer
 244  000000FC 1280                      move.b  d0,(a1)                 set pctl, requesting the first word in
 245
 246  000000FE 136E 0014 d_enab          move.b  enab(a6),3(a1)          enable the gpio card for dma
               0003
 247
 248  00000104 246E 0010                 movea.l chan(a6),a2             dma channel base address
 249  00000108 24AE 000C                 move.l  buf(a6),(a2)            set the dma address
 250  0000010C 357C FFFF                 move    #-1,4(a2)               set count to -1 for the case of one transfer
               0004
 251  00000112 202E 0008                 move.l  len(a6),d0              transfer length
 252  00000116 5580                      subq.l  #2,d0                   length-2
 253  00000118 6D0A                      blt.s   di_reni                 branch if one transfer only
 254  0000011A 3540 0004                 move    d0,4(a2)                set the dma count
 255  0000011E 357C 0002                 move    #$0002,6(a2)            arm the dma channel
               0006
 256
 257  00000124 46DF      di_reni         move    (sp)+,sr                re-enable interrupts ******************
 258
 259  00000126 47E9 0007                 lea     7(a1),a3                gpio register 7 address
 260  0000012A 7003                      moveq   #3,d0                   psts bit (gpio register 7)
 261  0000012C 49EA 0007                 lea     7(a2),a4                dma status lower byte address
 262  00000130 7200                      moveq   #0,d1                   armed bit (dma status register)
 263
 264  00000132 0113      di_loop         btst    d0,(a3)                 psts?
 265  00000134 6600 FECC                 bne     ioresc_catchall         ioresc(zcatchall) if so
 266  00000138 0314                      btst    d1,(a4)                 dma channel still armed?
 267  0000013A 66F6                      bne     di_loop                 keep looping if so
 268
```



```
 270
 271  0000013C 0C6A FFFF                 cmpi    #-1,4(a2)               dma transfer complete normally?
               0004
 272  00000142 6600 FEC8                 bne     bus_error               branch if not (bus error)
 273
 274  00000146 6100 FECC                 bsr     waitready               wait for last handshake to complete
 275
 276  0000014A 206E 000C                 movea.l buf(a6),a0              buffer address
 277  0000014E 202E 0008                 move.l  len(a6),d0              transfer length in words
 278  00000152 D080                      add.l   d0,d0                   transfer length in bytes
 279  00000154 31A9 0004                 move    4(a1),-2(a0,d0.l)       transfer last word
               08FE
 280
 281  0000015A 4E5E                      unlk    a6                      remove our stack frame
 282  0000015C 205F                      movea.l (sp)+,a0                pop the return address
 283  0000015E DEFC 0014                 adda    #20,sp                  pop off the parameters
 284  00000162 4ED0                      jmp     (a0)                    return
 285
 286
 287                                    end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

MATCH

Purpose

MATCH matches symbols in an EXT table to a DEF table.

Usage

MATCH is used by the linking loader.

```
PASS 1 COMPLETE, ERRORS: 0
    1                       *       procedure matchdefext
    2                       *               var resolved, matched: boolean          return flags
    3                       *                   matchmask: 0..255;                  to be or'd when matched
    4                       *               var def_table, ext_table:
    5                       *                       array[0.. ] of char;            symbol tables
    6                       *               var ext_list:  array[0.. ] of 0..65535  index list
    7                       *                   N:  shortint                        number of EXT records
    8                       *                   DEF_length: integer                 size of DEF table
    9
   10    00000000                   rorg    0
   11
   12                               def     matchdefext
   13
   14              0000 0001 offset         equ     1               offset field of a GVR
   15              0000 0002 gvr_flag       equ     2               bit in a GVR indicating absence
   16
   17              0000 0000 ext_str        equ     a0              pointer to EXT symbol
   18              0000 0000 return         equ     a0              storage for return address
   19              0000 0001 ext_base       equ     a1              address of EXT table
   20              0000 0002 list           equ     a2              address of index list
   21              0000 0003 def_str        equ     a3              pointer to string in DEF table
   22              0000 0004 gvr_ptr        equ     a4              pointer to GVR in DEF table
   23                        *global        equ     a5              reserved for PASCAL
   24                        *local         equ     a6              base of local workspace
   25                        *sp            equ     a7              stack pointer
   26
   27              0000 0000 cond           equ     d0              temporary save for condition codes
   28              0000 0001 len            equ     d1              length of string or GVR
   29              0000 0002 ext_count      equ     d2              number of EXT symbols to process
   30              0000 0003 def_count      equ     d3              bytes of DEF table left
   31              0000 0004 def_addr       equ     d4              address of symbol part of DEF record
   32              0000 0005 match_resolve  equ     d5              two booleans packed in a word
   33              0000 0006 ext_offset     equ     d6              relative pointer to an EXT symbol
   34              0000 0007 mask           equ     d7              match mask
   35
   36              0000 0000 matchdefext    equ *
   37    00000000 205F               movea.l (sp)+,return            return address
   38    00000002 261F               move.l  (sp)+,def_count         length of DEF table
   39    00000004 341F               move.w  (sp)+,ext_count         length of index list, pointer table
   40    00000006 245F               movea.l (sp)+,list              address of EXT index list
   41    00000008 225F               movea.l (sp)+,ext_base          address of EXT table
   42    0000000A 285F               movea.l (sp)+,gvr_ptr           address of DEF symbol
   43    0000000C 3E1F               move.w  (sp)+,mask              flag pattern for matched GVR
   44    0000000E 2F08               move.l  return,-(sp)            replace return address
   45
   46    00000010 7A01               moveq   #$0001,match_resolve    matched := false ; resolved := true
   47
   48    00000012 5342               subq.w  #1,ext_count            number of EXT symbols to process
   49    00000014 657E               bcs.s   done                    all done if no EXT table
   50
   51    00000016 3C1A       L1      move.w  (list)+,ext_offset      get relative address of an EXT symbol
   52    00000018 56CA FFFC          dbne    ext_count,L1            scan till non-zero, meaning unresolved
   53    0000001C 6776               beq.s   done                    done if none
   54
   55    0000001E 4A83               tst.l   def_count               length of DEF symbol table
   56    00000020 6748               beq.s   no_defs                 done if none
   57
   58    00000022 280C       get_def move.l  gvr_ptr,def_addr        save address of DEF record
```



```
   59    00000024 4281               clr.l   len                     find GVR of DEF record
   60    00000026 1214               move.b  (gvr_ptr),len           length of string
   61    00000028 5441               addq.w  #2,len                  compute length of string part
   62    0000002A 0881 0000          bclr    #0,len
   63    0000002E D8C1               adda.w  len,gvr_ptr             skip over string
   64    00000030 9681               sub.l   len,def_count
   65    00000032 0814 0002          btst    #gvr_flag,(gvr_ptr)     check for exported symbol
   66    00000036 6626               bne.s   next_def                ignore if not present
   67
   68    00000038 41F1 6000 compare  lea     0(ext_base,ext_offset.w),ext_str  get addresses of two strings
   69    0000003C 2644               movea.l def_addr,def_str
   70
   71    0000003E 4241               clr.w   len                     standard string compare
   72    00000040 1218               move.b  (ext_str)+,len          get EXT symbol length
   73    00000042 B21B               cmp.b   (def_str)+,len          compare two lengths
   74    00000044 40C0               move    sr,cond                 save condition codes for later
   75    00000046 6304               bls.s   str_comp
   76    00000048 122B FFFF          move.b  -1(def_str),len         get minimum of two string lengths
   77    0000004C 5301       str_comp subq.b #1,len                  loop if at least one character
   78    0000004E 6508               bcs.s   cmp_end
   79    00000050 B10B       cmp_lp  cmpm.b  (def_str)+,(ext_str)+   compare string bodies
   80    00000052 56C9 FFFC          dbne    len,cmp_lp              loop till not equal or end of string
   81    00000056 6604               bne.s   nomatch                 if string bodies are equal
   82    00000058 44C0       cmp_end move    cond,ccr                        then compare lengths
   83
   84    0000005A 6712               beq.s   match                   three way branch on comparison
   85    0000005C 632A       nomatch bls.s   ext_low
   86
   87    0000005E 4281       next_def clr.l  len                     DEF does not match an EXT
   88    00000060 122C 0001          move.b  offset(gvr_ptr),len     get length of GVR
   89    00000064 D8C1               adda.w  len,gvr_ptr
   90    00000066 9681               sub.l   len,def_count           decrease size of DEF table to search
   91    00000068 62B8               bhi.s   get_def                 loop back if there is any left
   92
   93    0000006A 4205       no_defs clr.b   match_resolve           resolved := false
   94    0000006C 6026               bra.s   done                    no more DEF records
   95
   96    0000006E 0045 0100 match    ori.w   #$0100,match_resolve    matched := true
   97    00000072 2384 6000          move.l  def_addr,0(ext_base,ext_offset.w)   copy address of DEF to EXT
   98    00000076 8F14               or.b    mask,(gvr_ptr)          mark DEF record as having been matched
   99    00000078 426A FFFE          clr.w   -2(list)                index := 0 to flag it as resolved
  100    0000007C 6002               bra.s   nxtext2                 find next unresolved EXT
  101    0000007E 3C1A       L2      move.w  (list)+,ext_offset      get relative address
  102    00000080 56CA FFFC nxtext2 dbne     ext_count,L2            scan till non zero or end of list
  103    00000084 66D8               bne.s   next_def                find next DEF record
  104    00000086 600C               bra.s   done                    done if none
  105
  106    00000088 4205       ext_low clr.b   match_resolve           resolved := false
  107    0000008A 6002               bra.s   nxtext3                 find next unresolved EXT
  108    0000008C 3C1A       L3      move.w  (list)+,ext_offset      get relative address
  109    0000008E 56CA FFFC nxtext3 dbne     ext_count,L3            scan till non zero or end of list
  110    00000092 66A4               bne.s   compare                 compare to same DEF record
  111
  112    00000094 205F       done    movea.l (sp)+,return            return address
  113    00000096 225F               movea.l (sp)+,a1                address of matched
  114    00000098 245F               movea.l (sp)+,a2                address of resolved
  115    0000009A 1485               move.b  match_resolve,(a2)      return boolean results
```

```
  116    0000009C E04D              lsr.w   #8,match_resolve
  117    0000009E 8B11              or.b    match_resolve,(a1)      matched := matched or match
  118    000000A0 4ED0              jmp     (return)
  119
  120                               end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

*** NO EXTERNAL SYMBOLS ***

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
CCR	STREG	0		00000005
CMP_END	REL	82		00000058
CMP_LP	REL	79		00000050
COMPARE	REL	68		00000038
COND	DREG	27		00000000
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DEF_ADDR	DREG	31		00000004
DEF_COUNT	DREG	30		00000003
DEF_STR	AREG	21		00000003
DONE	REL	112		00000094
EXT_BASE	AREG	19		00000001
EXT_COUNT	DREG	29		00000002
EXT_LOW	REL	106		00000088
EXT_OFFSET	DREG	33		00000006
EXT_STR	AREG	17		00000000
GET_DEF	REL	58		00000022
GVR_FLAG	ABS	15		00000002
GVR_PTR	AREG	22		00000004
L1	REL	51		00000016
L2	REL	101		0000007E
L3	REL	108		0000008C
LEN	DREG	28		00000001
LIST	AREG	20		00000002
MASK	DREG	34		00000007
MATCH	REL	96		0000006E
MATCHDEFEXT	REL	36		00000000
MATCH_RESOLVE	DREG	32		00000005
NEXT_DEF	REL	87		0000005E
NOMATCH	REL	85		0000005C
NO_DEFS	REL	93		0000006A
NXTEXT2	REL	102		00000080
NXTEXT3	REL	109		0000008E

```
OFFSET          ABS       14                    00000001
RETURN          AREG      18                    00000000
SP              AREG       0                    00000007
SR              STREG      0                    00000006
STR_COMP        REL       77                    0000004C
USP             STREG      0                    00000007
```

MATCHSTR

Purpose

MATCHSTR contains low-level assembly language string functions

```
PAGE   1 [2.0] 11/4/82 14:26:47  ASSEMBLY OF MATCHSTR.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                         * match pack
    2                         *
    3                                 nosyms
    4                                 sprint
    5                                 mname   matchstr
    6
    7                                 src     module matchstr;
    8                                 src     export
    9                                 src
   10                                 src     type
   11                                 src       stringarg=string[255];
   12                                 src       ttable   =packed array[0..0] of 0..255;
   13                                 src
   14                                 src     function afterstr(var s1:string;
   15                                 src                               c :integer;
   16                                 src                               n :integer;
   17                                 src                               s2:stringarg):integer;
   18                                 src     function beforestr(var s1:string;
   19                                 src                                c :integer;
   20                                 src                                n :integer;
   21                                 src                                s2:stringarg):integer;
   22                                 src     function changestr(var s1:string;
   23                                 src                                c :integer;
   24                                 src                                n :integer;
   25                                 src                                s2:stringarg;
   26                                 src                                s3:stringarg):integer;
   27                                 src     function breakstr(s1:stringarg;
   28                                 src                           c :integer;
   29                                 src                          s2:stringarg):integer;
   30                                 src     function spanstr(s1:stringarg;
   31                                 src                           c :integer;
   32                                 src                           s2:stringarg):integer;
   33                                 src     end;
   34
   35                                 def     matchstr_matchstr
   36              0000 0000 matchstr_matchstr     equ     *
   37     00000000 4E75                 rts
   38              0000 001A afunc     equ     26
   39              0000 0018 ams1      equ     24
   40              0000 0014 as1       equ     20
   41              0000 0010 ac        equ     16
   42              0000 000C an        equ     12
   43              0000 0008 as2       equ     8
   44              0000 0012 aargs     equ     18
   45                                 def     matchstr_afterstr
   46     00000002 0000                 dc.w    0
   47              0000 0004 matchstr_afterstr  equ     *
   48     00000004 4E56 0000            link    a6,#0
   49     00000008 42AE 001A            clr.l   afunc(a6)       func:=0
   50     0000000C 206E 0014            movea.l as1(a6),a0      a0:=^s1
   51     00000010 7000                 moveq   #0,d0
   52     00000012 1010                 move.b  (a0),d0         d0:=strlen(s1)
   53     00000014 242E 000C            move.l  an(a6),d2     if n>=0
   54     00000018 6D24                 blt.s   after1        then
   55                               *
   56                               * count
   57                               *
   58     0000001A 262E 0010            move.l  ac(a6),d3       d3:=c
```

```
PAGE   2 [2.0] 11/4/82 14:26:47  ASSEMBLY OF MATCHSTR.TEXT

   59     0000001E 6F2E                 ble.s   aret    check cursor<=0
   60     00000020 226E 0008            movea.l as2(a6),a1      a1:=^s1
   61     00000024 7200                 moveq   #0,d1
   62     00000026 1219                 move.b  (a1)+,d1        d1:=strlen(s2)
   63     00000028 6708                 beq.s   after0
   64     0000002A 6144                 bsr.s   scan
   65     0000002C 2D43 001A agood      move.l  d3,afunc(a6)
   66     00000030 601C                 bra.s   aret
   67                               *
   68                               * count but no string
   69                               *
   70     00000032 D682         after0  add.l   d2,d3   c:=c+n
   71     00000034 9083                 sub.l   d3,d0
   72     00000036 5280                 addq.l  #1,d0   c :: len(s1)+1
   73     00000038 66F2                 bne     agood
   74     0000003A 6000 0012            bra     aret
   75                               *                                 else
   76                               * no count given
   77                               *
   78     0000003E 226E 0008 after1   movea.l as2(a6),a1
   79     00000042 7200               moveq   #0,d1
   80     00000044 1219               move.b  (a1)+,d1
   81     00000046 6610               bne.s   after2
   82                               *
   83                               * no count no string
   84                               *
   85     00000048 5280                 addq.l  #1,d0           func:=d0+1
   86     0000004A 2D40 001A            move.l  d0,afunc(a6)
   87
   88     0000004E 4E5E         aret    unlk    a6              end
   89     00000050 205F                 movea.l (sp)+,a0
   90     00000052 DEFC 0012            adda.w  #aargs,sp
   91     00000056 4ED0                 jmp     (a0)
   92                               *
   93                               * no count with string
   94                               *
   95     00000058 7401         after2  moveq   #1,d2   count 1
   96     0000005A 262E 0010            move.l  ac(a6),d3
   97     0000005E 6FEE                 ble     aret    cursor out of range?
   98     00000060 610E                 bsr.s   scan
   99     00000062 67EA                 beq     aret    must match at least once
  100     00000064 2D43 001A after3   move.l  d3,afunc(a6)
  101     00000068 7401               moveq   #1,d2   reset count
  102     0000006A 6104               bsr.s   scan
  103     0000006C 66F6               bne     after3
  104     0000006E 60DE               bra     aret
  105                               *
  106              0000 0070 scan     equ     *
  107     00000070 5382               subq.l  #1,d2   pre decriment count
  108                               *
  109     00000072 48E7 E0C0 scan1    movem.l d0-d2/a0-a1,-(sp) save for next call
  110     00000076 6100 000E          bsr     scanloop
  111     0000007A 4CDF 0307          movem.l (sp)+,d0-d2/a0-a1
  112     0000007E 6704               beq.s   scanx
  113     00000080 51CA FFF0          dbra    d2,scan1
  114     00000084 4E75         scanx   rts
  115              0000 0086 scanloop        equ     *
```

```
 116    00000086 9083              sub.l   d3,d0
 117    00000088 5280              addq.l  #1,d0
 118    0000008A 6D00 0036         blt     sfexit  pos in range ?
 119
 120    0000008E 9041              sub     d1,d0   is str2 longer than
 121    00000090 6D00 0030         blt     sfexit  remaining str1 ?
 122
 123    00000094 4A41              tst.w   d1
 124    00000096 6726              beq.s   ssexit2 str2 is null so match
 125
 126    00000098 2448              movea.l a0,a2   save str1 ptr
 127    0000009A D0C3              adda.w  d3,a0   start source compare
 128
 129    0000009C 1C19              move.b  (a1)+,d6        first character
 130    0000009E 5541              subq    #2,d1
 131
 132    000000A0 BC18      scl1    cmp.b   (a0)+,d6
 133    000000A2 57C8 FFFC scl2    dbeq    d0,scl1
 134    000000A6 661A              bne.s   sfexit  found it ?
 135
 136    000000A8 2648              movea.l a0,a3   temp str1
 137    000000AA 2849              movea.l a1,a4   temp str2
 138
 139    000000AC 3A01              move.w  d1,d5   remaining str2 bytes
 140    000000AE 6D08              blt.s   ssexit  str2 is 1 char
 141
 142    000000B0 B90B      scl3    cmpm.b  (a3)+,(a4)+
 143    000000B2 56CD FFFC         dbne    d5,scl3
 144    000000B6 66EA              bne     scl2
 145    000000B8 260B      ssexit  move.l  a3,d3
 146    000000BA 968A              sub.l   a2,d3
 147    000000BC 4E75              rts
 148    000000BE 2603      ssexit2 move.l  d3,d3   set condition code
 149    000000C0 4E75              rts
 150
 151    000000C2 7600      sfexit  moveq   #0,d3   cursor to zero
 152    000000C4 4E75              rts
 153                       *
 154                               def     matchstr_beforestr
 155    000000C6 0000              dc.w    0
 156              0000 00C8 matchstr_beforestr equ     *
 157    000000C8 4E56 0000         link    a6,#0
 158    000000CC 42AE 001A         clr.l   afunc(a6)       func:=0
 159    000000D0 206E 0014         movea.l as1(a6),a0      a0:=^s1
 160    000000D4 7000              moveq   #0,d0
 161    000000D6 1010              move.b  (a0),d0         d0:=strlen(s1)
 162    000000D8 242E 000C         move.l  an(a6),d2    if n>=0
 163    000000DC 6D2A              blt.s   before1      then
 164                       *
 165                       * count
 166                       *
 167    000000DE 262E 0010         move.l  ac(a6),d3       d3:=c
 168    000000E2 6FC0 FF6A         ble     aret    check cursor<=0
 169    000000E6 226E 0008         movea.l as2(a6),a1      a1:=^s1
 170    000000EA 7200              moveq   #0,d1
 171    000000EC 1219              move.b  (a1)+,d1        d1:=strlen(s2)
 172    000000EE 6710              beq.s   before0
```



```
 173    000000F0 6100 FF7E         bsr     scan
 174    000000F4 6702              beq.s   bgood
 175    000000F6 9681              sub.l   d1,d3   move to front of match
 176    000000F8 2D43 001A bgood   move.l  d3,afunc(a6)
 177    000000FC 6000 FF50         bra     aret
 178                       *
 179                       * count but no string
 180                       *
 181    00000100 9682      before0 sub.l   d2,d3           c:=c-n
 182    00000102 6F00 FF4A         ble     aret    c :: 0
 183    00000106 60F0              bra     bgood
 184                       *                               else
 185                       * no count given
 186                       *
 187    00000108 226E 0008 before1 movea.l as2(a6),a1
 188    0000010C 7200              moveq   #0,d1
 189    0000010E 1219              move.b  (a1)+,d1
 190    00000110 660A              bne.s   before2
 191                       *
 192                       * no count no string
 193                       *
 194    00000112 7001              moveq   #1,d0           func:=1
 195    00000114 2D40 001A         move.l  d0,afunc(a6)
 196    00000118 6000 FF34         bra     aret
 197                       *
 198                       * no count with string
 199                       *
 200    0000011C 7401      before2 moveq   #1,d2   count 1
 201    0000011E 262E 0010         move.l  ac(a6),d3
 202    00000122 6F00 FF2A         ble     aret    cursor out of range?
 203    00000126 6100 FF48         bsr     scan
 204    0000012A 6700 FF22         beq     aret    must match at least once
 205    0000012E 2D43 001A before3 move.l  d3,afunc(a6)
 206    00000132 93AE 001A         sub.l   d1,afunc(a6)    move to front of match
 207    00000136 7401              moveq   #1,d2   reset count
 208    00000138 6100 FF36         bsr     scan
 209    0000013C 66F0              bne     before3
 210    0000013E 6000 FF0E         bra     aret
 211
 212              0000 001E mf      equ     30
 213              0000 001C ms1m    equ     28
 214              0000 0018 ms1     equ     24
 215              0000 0014 mc      equ     20
 216              0000 0010 mk      equ     16
 217              0000 000C ms2     equ     12
 218              0000 0008 ms3     equ     8
 219              0000 0016 mr      equ     22
 220                               def     matchstr_changestr
 221    00000142 0000              dc.w    0
 222              0000 0144 matchstr_changestr equ     *
 223    00000144 4E56 0000         link    a6,#0
 224
 225    00000148 42AE 001E         clr.l   mf(a6)  function result 0
 226
 227    0000014C 282E 0014         move.l  mc(a6),d4       cursor
 228    00000150 6F00 0166         ble     chgret
 229
```

```
00000154 206E 0018          movea.l ms1(a6),a0
00000158 7000               moveq   #0,d0
0000015A 1010               move.b  (a0),d0
0000015C 9084               sub.l   d4,d0
0000015E 5280               addq.l  #1,d0
00000160 6D00 0156          blt     chgret  cursor in range ?

00000164 2A2E 0010          move.l  mk(a6),d5       counter
00000168 6700 0046          beq     chgzcnt
0000016C 6F00 0054          ble     chgncnt

                      *     have count value

00000170 226E 000C          movea.l ms2(a6),a1
00000174 4A11               tst.b   (a1)
00000176 6700 0082          beq     chgcnil1

                      *     have count and s2 and maybe s3
                      *     replace the next n occurences of s2 with s3

0000017A 6014               bra.s   chgl2
0000017C 48E7 0400 chgl1    movem.l d5,-(sp)        save count
00000180 6100 00AE          bsr     chgflds
00000184 4CDF 0020          movem.l (sp)+,d5        get count
00000188 2D44 001E          move.l  d4,mf(a6)       set func
0000018C 6700 012A          beq     chgret
00000190 51CD FFEA chgl2    dbra    d5,chgl1
00000194 6000 0122          bra     chgret

                      *     count but no s2
                      *     replace next count chars with s3
00000198 7000               moveq   #0,d0
0000019A 1010               move.b  (a0),d0
0000019C 2600               move.l  d0,d3   final length of s1
0000019E 9685               sub.l   d5,d3
000001A0 246E 0008          movea.l ms3(a6),a2
000001A4 7400               moveq   #0,d2
000001A6 1412               move.b  (a2),d2
000001A8 D682               add.l   d2,d3
000001AA 6D00 010C          blt     chgret  count is too big

                      *     count is zero
                      *     insert s3 at cursor

000001AE 0000       nilstr  dc.w    0
                      *     make ms2 a dummy nilstring
000001B0 2D7C 0000 chgzcnt move.l   #nilstr,ms2(a6)
         01AE 000C
000001B8 6176               bsr.s   chgflds
000001BA 2D44 001E          move.l  d4,mf(a6)
000001BE 6000 00F8          bra     chgret
                      *
                      *     no count
                      *
000001C2 226E 000C chgncnt  movea.l ms2(a6),a1
000001C6 4A11               tst.b   (a1)
000001C8 6716               beq.s   chgnil1
```



```
                      *     no count but has s2 might have s3
                      *     replace all occurences of s2 with s3

000001CA 6164               bsr.s   chgflds
000001CC 2D44 001E          move.l  d4,mf(a6)
000001D0 6700 00E6          beq     chgret  must change at least one
000001D4 615A       chgncnt1 bsr.s  chgflds
000001D6 2D44 001E          move.l  d4,mf(a6)       set func value
000001DA 6700 00DC          beq     chgret
000001DE 60F4               bra     chgncnt1

                      *     no count no s2

000001E0 246E 0008 chgnil1  movea.l ms3(a6),a2
000001E4 4A12               tst.b   (a2)
000001E6 6706               beq.s   chgnil2

                      *     no count only s3
                      *     replace rest of s1 with s3

000001E8 1084               move.b  d4,(a0) chop s1 to cursor
000001EA 5310               subq.b  #1,(a0)
000001EC 60C2               bra     chgzcnt add s3
                      *
                      * no count no strings
                      *     delete remainder of s1

000001EE 1084       chgnil2 move.b  d4,(a0) set s1 length
000001F0 5310               subq.b  #1,(a0)
000001F2 2D44 001E          move.l  d4,mf(a6) set func value
000001F6 6000 00C0          bra     chgret

                      *     have count no s2
                      *     replace count bytes with s3

000001FA 9085       chgcnil1 sub.l  d5,d0   d0 is #bytes after delete
000001FC 6D00 00BA          blt     chgret

00000200 246E 0008          movea.l ms3(a6),a2 addr and size of s3
00000204 7400               moveq   #0,d2
00000206 141A               move.b  (a2)+,d2

00000208 7600               moveq   #0,d3
0000020A 1610               move.b  (a0),d3 will it fit
0000020C 9645               sub     d5,d3
0000020E D642               add     d2,d3   final size of s1
00000210 B62E 001C          cmp.b   ms1m(a6),d3
00000214 6200 00A2          bhi     chgret

00000218 2E05               move.l  d5,d7   apparent size of s2

0000021A D0C4               adda    d4,a0   cursor addr
0000021C 47F0 5000          lea     0(a0,d5.w),a3   after delete
00000220 6100 0064          bsr     chgf1   do it
00000224 2D44 001E          move.l  d4,mf(a6)
00000228 6000 008E          bra     chgret
```

```
0000022C 7800      chgbad  moveq   #0,d4   cursor to zero
0000022E 4E75              rts
                   *
                   *       do one change
                   *
00000230 206E 0018 chgflds movea.l ms1(a6),a0     source string
00000234 7000              moveq   #0,d0
00000236 1010              move.b  (a0),d0        s length
00000238 2600              move.l  d0,d3   final length of s
0000023A 9084              sub.l   d4,d0
0000023C 5280              addq.l  #1,d0
0000023E 6DEC              blt     chgbad  pos in range ?

00000240 226E 000C         movea.l ms2(a6),a1      old string
00000244 7200              moveq   #0,d1
00000246 1219              move.b  (a1)+,d1        old lenght
00000248 2E01              move.l  d1,d7   save it for later

0000024A 246E 0008         movea.l ms3(a6),a2      new string
0000024E 7400              moveq   #0,d2
00000250 141A              move.b  (a2)+,d2        new length

00000252 9641              sub     d1,d3
00000254 D642              add     d2,d3
00000256 B62E 001C         cmp.b   msim(a6),d3     will it all fit ?
0000025A 62D0              bhi     chgbad

0000025C D0C4              adda.w  d4,a0   start source compare

0000025E 4A01              tst.b   d1
00000260 6700 006E         beq     chgins          0 length so match

00000264 9041              sub     d1,d0   is old longer than
00000266 6DC4              blt     chgbad  remaining source ?

00000268 1C19              move.b  (a1)+,d6        first character
0000026A 5541              subq    #2,d1

0000026C BC18      chg1    cmp.b   (a0)+,d6
0000026E 57C8 FFFC chg2    dbeq    d0,chg1
00000272 6644              bne.s   chgret  found it ?

00000274 2648              movea.l a0,a3   temp source
00000276 2849              movea.l a1,a4   temp old

00000278 3A01              move.w  d1,d5   remaining old bytes
0000027A 6D08              blt.s   chgf0   old is 1 char

0000027C B90B      chg3    cmpm.b  (a3)+,(a4)+
0000027E 56CD FFFC         dbne    d5,chg3
00000282 66EA              bne     chg2

00000284 5388      chgf0   subq.l  #1,a0

00000286 286E 0018 chgf1   movea.l ms1(a6),a4       string s
0000028A 1883              move.b  d3,(a4)
```



```
0000028C 4A40              tst.w   d0
0000028E 6700 001C         beq     chgcpy1
00000292 9E42              sub.w   d2,d7
00000294 6716              beq.s   chgcpy1
00000296 6E2A              bgt.s   chgsml
                   * new string is greater than old
00000298 49F4 3001         lea     1(a4,d3.w),a4     end s + 1
0000029C 47F4 7000         lea     0(a4,d7.w),a3   source
000002A0 5340              subq.w  #1,d0   count

000002A2 1923      chgins1 move.b  -(a3),-(a4)
000002A4 51C8 FFFC         dbra    d0,chgins1
000002A8 6002              bra.s   chgcpy1

000002AA 10DA      chgcpy  move.b  (a2)+,(a0)+
000002AC 51CA FFFC chgcpy1 dbra    d2,chgcpy

000002B0 91EE 0018         suba.l  ms1(a6),a0
000002B4 2808              move.l  a0,d4           new cursor value
000002B6 4E75              rts

000002B8 4E5E      chgret  unlk    a6
000002BA 205F              movea.l (sp)+,a0
000002BC DEFC 0016         adda.w  #mr,sp
000002C0 4ED0              jmp     (a0)
                   *
                   * new string is smaller than old
           0000 02C2 chgsml  equ     *
000002C2 284B              movea.l a3,a4
000002C4 98C7              suba.w  d7,a4
000002C6 5340              subq.w  #1,d0

000002C8 18DB      chgdel1 move.b  (a3)+,(a4)+
000002CA 51C8 FFFC         dbra    d0,chgdel1
000002CE 60DC              bra     chgcpy1

           0000 02D0 chgins  equ     *
000002D0 2648              movea.l a0,a3
000002D2 60B2              bra     chgf1

           0000 0014 bf      equ     20
           0000 0010 bs1     equ     16
           0000 000C bc      equ     12
           0000 0008 bs2     equ     8
           0000 000C bargs   equ     12

                           def     matchstr_breakstr
000002D4 0000              dc.w    0
           0000 02D6 matchstr_breakstr  equ     *
000002D6 4E56 0000         link    a6,#0
000002DA 42AE 0014         clr.l   bf(a6)          set func to 0

000002DE 282E 000C         move.l  bc(a6),d4       cursor pos
000002E2 6F3A              ble.s   bsret

000002E4 206E 0010         movea.l bs1(a6),a0
000002E8 2848              movea.l a0,a4   save addr of s1
```

```
PAGE   9 [2.0] 11/4/82 14:26:47  ASSEMBLY OF MATCHSTR.TEXT

  457    000002EA 7000              moveq   #0,d0
  458    000002EC 1010              move.b  (a0),d0       length s1
  459    000002EE 672E              beq.s   bsret
  460
  461    000002F0 9084              sub.l   d4,d0
  462    000002F2 6D2A              blt.s   bsret
  463
  464    000002F4 226E 0008         movea.l bs2(a6),a1    list addr
  465    000002F8 7200              moveq   #0,d1
  466    000002FA 1219              move.b  (a1)+,d1      list length
  467    000002FC 6720              beq.s   bsret
  468
  469    000002FE D1C4              adda.l  d4,a0   start scan
  470    00000300 5341              subq.w  #1,d1
  471
  472    00000302 1418      bloop0  move.b  (a0)+,d2        char to test
  473    00000304 2449              movea.l a1,a2   copy list addr
  474    00000306 3601              move.w  d1,d3   copy list length
  475
  476    00000308 B41A      bloop1  cmp.b   (a2)+,d2
  477    0000030A 57CB FFFC         dbeq    d3,bloop1
  478    0000030E 6706              beq.s   bsxit
  479
  480    00000310 51C8 FFF0         dbra    d0,bloop0
  481    00000314 6008              bra.s   bsret
  482
  483    00000316 91CC      bsxit   suba.l  a4,a0   calc func value
  484    00000318 5388              subq.l  #1,a0
  485    0000031A 2D48 0014         move.l  a0,bf(a6)
  486
  487    0000031E 4E5E      bsret   unlk    a6
  488    00000320 205F              movea.l (sp)+,a0
  489    00000322 DEFC 000C         adda.w  #bargs,sp
  490    00000326 4ED0              jmp     (a0)
  491
  492                               def     matchstr_spanstr
  493    00000328 0000              dc.w    0
  494              0000 032A matchstr_spanstr   equ     *
  495    0000032A 4E56 0000         link    a6,#0
  496    0000032E 42AE 0014         clr.l   bf(a6)  zero function value
  497    00000332 282E 000C         move.l  bc(a6),d4       cursor position
  498    00000336 6FE6              ble.s   bsret
  499
  500    00000338 206E 0010         movea.l bs1(a6),a0      string addr
  501    0000033C 2848              movea.l a0,a4
  502    0000033E 7000              moveq   #0,d0
  503    00000340 1010              move.b  (a0),d0         string length
  504    00000342 67DA              beq.s   bsret
  505    00000344 9084              sub.l   d4,d0
  506    00000346 6DD6              blt.s   bsret
  507
  508    00000348 226E 0008         movea.l bs2(a6),a1      list addr
  509    0000034C 7200              moveq   #0,d1
  510    0000034E 1219              move.b  (a1)+,d1        list length
  511    00000350 67CC              beq.s   bsret
  512
  513    00000352 D1C4              adda.l  d4,a0   start scan
```

```
PAGE  10 [2.0] 11/4/82 14:26:47  ASSEMBLY OF MATCHSTR.TEXT

  514    00000354 5341              subq.w  #1,d1
  515
  516    00000356 1418      sloop0  move.b  (a0)+,d2
  517    00000358 2449              movea.l a1,a2   copy list addr
  518    0000035A 2601              move.l  d1,d3   copy list length
  519
  520    0000035C B41A      sloop1  cmp.b   (a2)+,d2
  521    0000035E 57CB FFFC         dbeq    d3,sloop1
  522    00000362 66B2              bne     bsxit
  523
  524    00000364 51C8 FFF0         dbra    d0,sloop0
  525    00000368 60AC              bra     bsxit
  526
  527                               end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

MODIV

Purpose

MODIV provides the 32-bit integer MOD and DIV functions which conform to the Pascal definition.

```
PASS 1 COMPLETE. ERRORS: 0
    1                       *----------------------------------------
    2                        DEF ASM_DIV
    3                        DEF ASM_MOD
    4                       *----------------------------------------
    5                       *               register usage
    6                       *               d0 - return address
    7                       *               d1 - divisor
    8                       *               d2 - dividend and quotient
    9                       *               d3 - remainder
   10                       *               d4 - loop counter
   11                       *               d5 - sign of remainder
   12                       *               d6 - sign of quotent
   13                       *               d7 - mod/div flag
   14                       *
   15    00000000 4E44      ovflow          trap            #4
   16    00000002 4E47      valrange        trap            #7
   17    00000004 7E01      asm_mod         moveq           #1,d7           set mod flag
   18    00000006 4CDF 0007                 movem.l         (sp)+,d0/d1/d2  read return addr and operands
   19    0000000A 4A81                      tst.l           d1              divide by zero?
   20    0000000C 6A0A                      bpl.s           d_start         for mod?
   21    0000000E 60F2                      bra.s           valrange
   22    00000010 7E00      asm_div         moveq           #0,d7           clear mod flag
   23    00000012 4CDF 0007                 movem.l         (sp)+,d0/d1/d2  read return addr and operands
   24    00000016 4A81                      tst.l           d1              divide by zero?
   25    00000018 6720      d_start         beq.s           zerodiv
   26    0000001A 3041                      movea.w         d1,a0           is divisor a
   27    0000001C B288                      cmp.l           a0,d1           16 bit integer?
   28    0000001E 661C                      bne.s           do_full
   29    00000020 2602                      move.l          d2,d3           try signed divide
   30    00000022 87C1                      divs            d1,d3
   31    00000024 6916                      bvs.s           do_full         did it work?
   32    00000026 4A47                      tst.w           d7              mod or div?
   33    00000028 6708                      beq.s           div_1
   34    0000002A 4843                      swap            d3
   35    0000002C 4A43                      tst.w           d3              if mod is negative
   36    0000002E 6A02                      bpl.s           div_1           teen
   37    00000030 D641                      add.w           d1,d3           add back divisor
   38    00000032 48C3      div_1           ext.l           d3
   39    00000034 2F03      dm_out          move.l          d3,-(sp)        push result
   40    00000036 2040                      movea.l         d0,a0
   41    00000038 4ED0                      jmp             (a0)            fake return
   42    0000003A 4E45      zerodiv         trap            #5
   43                       *
   44                       *               convert divisor and dividend to sign magnitude
   45                       *
   46    0000003C 780F      do_full         moveq           #15,d4          loop count - 1
   47    0000003E 7C00                      moveq           #0,d6           sign of quotient
   48    00000040 7A00                      moveq           #0,d5           sign of remainder
   49    00000042 4A81                      tst.l           d1              divisor negative?
   50    00000044 6A06                      bpl.s           divend
   51    00000046 4481                      neg.l           d1
   52    00000048 6954                      bvs.s           max_neg_dvsr
   53    0000004A 4646                      not.w           d6              set sign flag
   54    0000004C 4A82      divend          tst.l           d2              dividend negative
   55    0000004E 6A14                      bpl.s           rmndr
   56    00000050 4482                      neg.l           d2              complement quotient sign
   57    00000052 680C                      bvc.s           not_special
```



```
   58    00000054 B2BC 0000                 cmp.l           #1,d1           test for minint div -1
               0001
   59    0000005A 6604                      bne.s           not_special
   60    0000005C 4A46                      tst.w           d6
   61    0000005E 66A0                      bne.s           ovflow
   62    00000060 4646      not_special     not             d6              flag
   63    00000062 4645                      not             d5              negative remainder
   64    00000064 7600      rmndr           moveq           #0,d3           clear remainder
   65    00000066 4841                      swap            d1              is divisor <= 16 bits
   66    00000068 4A41                      tst.w           d1
   67    0000006A 6644                      bne.s           big_div
   68    0000006C 4842                      swap            d2
   69    0000006E 4841                      swap            d1
   70    00000070 3602                      move.w          d2,d3           get high order dividend
   71    00000072 86C1                      divu            d1,d3           high part of divide
   72    00000074 3403                      move.w          d3,d2           high quotient to d2
   73    00000076 4842                      swap            d2
   74    00000078 3602                      move.w          d2,d3           divide low order
   75    0000007A 86C1                      divu            d1,d3           dividend by divisor
   76    0000007C 3403                      move.w          d3,d2           quotient in d2
   77    0000007E 4243                      clr.w           d3
   78    00000080 4843                      swap            d3              remainder in d3
   79                       *               put in correct sign for quotient and remainder
   80    00000082 4A46      dm_fixup        tst.w           d6
   81    00000084 6A02                      bpl.s           chk_rem
   82    00000086 4482                      neg.l           d2
   83    00000088 4A45      chk_rem         tst.w           d5
   84    0000008A 6A02                      bpl.s           dm_store
   85    0000008C 4483                      neg.l           d3
   86    0000008E 4A47      dm_store        tst.w           d7              div or mod?
   87    00000090 6604                      bne.s           mod_out
   88    00000092 2602                      move.l          d2,d3
   89    00000094 609E                      bra.s           dm_out
   90    00000096 4A83      mod_out         tst.l           d3              if negative mod
   91    00000098 6A9A                      bpl.s           dm_out          then
   92    0000009A D681                      add.l           d1,d3           then add back divisor
   93    0000009C 6096                      bra.s           dm_out
   94                       *
   95                       *               handle maximum negative divisor
   96                       *
   97    0000009E 4482      max_neg_dvsr    neg.l           d2
   98    000000A0 6908                      bvs.s           max_max         test for max neg dividend
   99    000000A2 2602                      move.l          d2,d3
  100    000000A4 4483                      neg.l           d3
  101    000000A6 7400                      moveq           #0,d2
  102    000000A8 60E4                      bra.s           dm_store
  103    000000AA 7401      max_max         moveq           #1,d2
  104    000000AC 7600                      moveq           #0,d3
  105    000000AE 60DE                      bra.s           dm_store
  106                       *
  107                       *               32 bit divisor
  108                       *
  109    000000B0 4841      big_div         swap            d1              restore divisor
  110    000000B2 4842                      swap            d2              move high order
  111    000000B4 3602                      move.w          d2,d3           dividend to remainder
  112    000000B6 4242                      clr.w           d2              shift dividend 16 bits left
  113    000000B8 9681                      sub.l           d1,d3           subtract divisor from rem.
```

```
 114    000000BA 2041                      movea.l         d1,a0           divisor in d1
 115    000000BC 4481                      neg.l           d1              minus divisor in a0
 116    000000BE C388                      exg             d1,a0
 117                        *
 118                        *               co-routine for negative remainder
 119                        *
 120    000000C0 D482       m_top          add.l           d2,d2           shift dividend and quotient
 121    000000C2 D783                      addx.l          d3,d3           shift remainder
 122    000000C4 D681                      add.l           d1,d3           add divisor
 123    000000C6 6A12                      bpl.s           p_bottom        remainder positive?
 124    000000C8 51CC FFF6 m_bottom        dbra            d4,m_top        loop 16 times
 125    000000CC D681                      add.l           d1,d3           restore remainder
 126    000000CE D482                      add.l           d2,d2           shift in last bit of quotient
 127    000000D0 60B0                      bra.s           dm_fixup
 128                        *
 129                        *               co-routine for positive remainder
 130                        *
 131    000000D2 D582       p_top          addx.l          d2,d2           shift dividend and quotient
 132    000000D4 D783                      addx.l          d3,d3           shift remainder
 133    000000D6 D688                      add.l           a0,d3           subtract divisor
 134    000000D8 6BEE                      bmi.s           m_bottom        remainder negative?
 135    000000DA 51CC FFF6 p_bottom        dbra            d4,p_top        loop 16 times
 136    000000DE D582                      addx.l          d2,d2           shift in last bit of quotient
 137    000000E0 60A0                      bra.s           dm_fixup
 138                                end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

*** NO EXTERNAL SYMBOLS ***

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ASM_DIV	REL	22		00000010
ASM_MOD	REL	17		00000004
BIG_DIV	REL	109		000000B0
CCR	STREG	0		00000005
CHK_REM	REL	83		00000088
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DIVEND	REL	54		0000004C
DIV_1	REL	38		00000032
DM_FIXUP	REL	80		00000082
DM_OUT	REL	39		00000034
DM_STORE	REL	86		0000008E
DO_FULL	REL	46		0000003C
D_START	REL	25		00000018
MAX_MAX	REL	103		000000AA
MAX_NEG_DVSR	REL	97		0000009E
MOD_OUT	REL	90		00000096
M_BOTTOM	REL	124		000000C8
M_TOP	REL	120		000000C0
NOT_SPECIAL	REL	62		00000060
OVFLOW	REL	15		00000000
P_BOTTOM	REL	135		000000DA
P_TOP	REL	131		000000D2
RMNDR	REL	64		00000064
SP	AREG	0		00000007
SR	STREG	0		00000006
USP	STREG	0		00000007
VALRANGE	REL	16		00000002
ZERODIV	REL	42		0000003A

NEWWORDS

Purpose

NEWWORDS provides the procedure which implements NEW (if $HEAPDISPOSE OFF$) called NEWBYTES, as well as procedure NEWWORDS and the function MEMAVAIL.

Usage

NEWWORDS calls are emitted by the compiler to these routines.

Notes

MEMAVAIL returns the amount of free memory in BYTES. (A WORD is 2-bytes or 16-bits.)

```
 PAGE   1 [2.0] 11/4/82 13:13:56  ASSEMBLY OF NEWWORDS.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                           *       function memavail: integer;     {return number of free bytes}
    2                           *       procedure newwords(var p: anyptr; size: integer);
    3                           *                       {implement the Pascal NEW( ) procedure}
    4
    5     00000000                      rorg 0
    6                                   def     asm_memavail,asm_newwords,asm_newbytes
    7                                   refa    stackfudge,sysglobals
    8                                   nosyms
    9
   10               FFFF FFF2 heapptr equ sysglobals-14       location of heap pointer
   11
   12               0000 0000 asm_memavail equ *
   13     00000000 205F                movea.l (SP)+,a0        return address
   14     00000002 200F                move.l  SP,d0           compute result = (SP)-(heap pointer)
   15     00000004 90AD FFF2           sub.l   heapptr(a5),d0
   16     00000008 90BC 0000           sub.l   #stackfudge,d0
                   0000
   17     0000000E 6C02                bge.s   ge
   18     00000010 4280                clr.l   d0
   19     00000012 2E80          ge    move.l  d0,(SP)
   20     00000014 4ED0                jmp     (a0)            rts
   21
   22               0000 0016 asm_newwords equ *
   23     00000016 205F                movea.l (SP)+,a0        return address
   24     00000018 201F                move.l  (SP)+,d0        size of requested allocation in words
   25     0000001A E380                asl.l   #1,d0           convert to bytes
   26     0000001C 4EFA 000C           jmp     alloc           same as newbytes from here on
   27
   28               0000 0020 asm_newbytes equ *
   29     00000020 205F                movea.l (SP)+,a0        return address
   30     00000022 201F                move.l  (SP)+,d0        size of requested allocation in bytes
   31     00000024 5280                addq.l  #1,d0           round up to an even number of bytes
   32     00000026 0880 0000           bclr    #0,d0
   33     0000002A 225F          alloc movea.l (SP)+,a1        address of pointer return variable
   34     0000002C 246D FFF2           movea.l heapptr(a5),a2  pointer := heapptr
   35     00000030 228A                move.l  a2,(a1)
   36     00000032 D5C0                adda.l  d0,a2           bump heap by size of new object
   37     00000034 47EF 0000           lea     -stackfudge(sp),a3
   38     00000038 B5CB                cmpa.l  a3,a2           check for heap overflow
   39     0000003A 6E06                bgt.s   heapover
   40     0000003C 2B4A FFF2           move.l  a2,heapptr(a5)  restore heap pointer
   41     00000040 4ED0                jmp     (a0)            rts
   42
   43     00000042 4E42          heapover trap #2              same trap as stack overflow
   44
   45                                  end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

POWERUP

Purpose

POWERUP provides INTERRUPT, TRAP and EXCEPTION handling (error recovery) as well as non-local GOTO and other miscellaneous utilities.

Usage

POWERUP is part of INITLIB.

Notes

Most TRAP and INTERRUPT vectors are initialized in POWERUP.

```
PASS 1 COMPLETE. ERRORS: 0
    1                           *       INTERRUPT, TRAP, AND EXCEPTION HANDLER
    2
    3                                   DEF     ASM_SETINTLEVEL,ASM_INTLEVEL
    4                                   DEF     ASM_INITVECTS,RESETX,ASM_CLOSEFILES
    5                                   DEF     DCRTINFO,EXCP_PC,EXCP_LINE
    6                                   DEF     GRAPHICSBASE,GRAPHICSFLAG,ALPHAFLAG
    7                                   DEF     ASM_IAND,ASM_IOR
    8
    9                                   REFA    SYSGLOBALS,LOADER,STACKFUDGE
   10                                   REFR    FS_FCLOSE,INITUNITS_NOISR,INITLOAD_INITLOAD
   11                                   LMODE   FS_FCLOSE,INITUNITS_NOISR,INITLOAD_INITLOAD
   12                                   SMODE   ESCAPE
   13
   14                          *THE FOLLOWING OFFSETS ARE RELATIVE TO SYSGLOBALS(A5)
   15
   16              FFFF FFFE ESCAPECODE        EQU SYSGLOBALS-2            (A5)
   17              FFFF FFFA FILELISTPTR       EQU SYSGLOBALS-6            (A5)
   18              FFFF FFF6 RECOVERBLOCK      EQU SYSGLOBALS-10           (A5)
   19              FFFF FFF2 HEAPPOINTER       EQU SYSGLOBALS-14           (A5)
   20              FFFF FFEE HEAPBASE          EQU SYSGLOBALS-18           (A5)
   21              FFFF FFEA IORESULT          EQU SYSGLOBALS-22           (A5)
   22              FFFF FFC2 INTERRUPTTABLE    EQU SYSGLOBALS-62           (A5) ADDRESS OF INTERRUPTTABLE[1..7]
   23              FFFF FFBE ENDISRHOOK        EQU SYSGLOBALS-66           (A5) ADDRESS OF END OF ISR ROUTINE
   24              FFFF FEEC DEBUGGER          EQU SYSGLOBALS-276          (A5) DEBUGGER HOOK
   25              FFFF FEE4 CLEARIOHOOK       EQU SYSGLOBALS-284          (A5) CLEAR I/O HOOK
   26              FFFF FFBA SYSDEFS           EQU LOADER-70               (A5) LINKED LIST OF PERMANT PROGRAMS
   27              0000 0000 NIL               EQU 0
   28
   29              FFFF FB00 HIGHMEM           EQU $FFFFFB00   LEAVES ROOM FOR VECTORS, MONITOR STUFF, ETC.
   30              FFFF FF94 TRAPOVECTOR       EQU $FFFFFF94   LOCATION OF EXCEPTION VECTOR FOR TRAP #0
   31                                          *
   32              FFFF FF9A LEVEL7V           EQU $FFFFFF9A   NMI
   33              FFFF FF34 KBDRESETV         EQU $FFFFFF34   KEY BOARD <SHIFT PAUSE VECTOR>
   34                                          *
   35              FFFF FDCE LOWMEM            EQU $FFFFFDCE   LOCATION IN BOOT ROM OF LOWEST RAM
   36              0000 01A0 RTN_TO_MONITOR    EQU $1A0        ENTRY POINT IN BOOT ROM
   37              0053 8000 G_OFF_MEM         EQU $538000     GRAPHICS OFF
   38              0051 2000 ALPHA_MEM         EQU $512000     ALPHA MEMORY
   39
   40              FFFF FB00 BUS_ERR_INFO      EQU HIGHMEM
   41              FFFF FB02 BUS_ERR_ADDR      EQU BUS_ERR_INFO+2
   42              FFFF FB06 BUS_ERR_INSTR     EQU BUS_ERR_ADDR+4
   43              FFFF FB08 EXCP_STATUS       EQU BUS_ERR_INSTR+2
   44              FFFF FB0A EXCP_PC           EQU EXCP_STATUS+2
   45              FFFF FB0E EXCP_LINE         EQU EXCP_PC+4
   46              FFFF FB12 LASTLINE          EQU EXCP_LINE+4
   47              FFFF FB16 ESCAPE            EQU LASTLINE+4
   48              FFFF FB1C PCTEMP            EQU ESCAPE+6
   49              FFFF FB20 SRTEMP            EQU PCTEMP+4
   50              FFFF FB22 INITSTACK         EQU SRTEMP+2
   51              FFFF FB26 INITPC            EQU INITSTACK+4
   52              FFFF FB2A INITRECOVER       EQU INITPC+4
   53              FFFF FB2E G_DOLLAR          EQU INITRECOVER+4
   54              FFFF FB32 CTL_RESETV        EQU G_DOLLAR+4          KEY BOARD <CONTROL><SHIFT><PAUSE> VECTOR
   55              FFFF FB38 DEBUGESCAPE       EQU CTL_RESETV+6
   56              FFFF FB3E DCRTINFO          EQU DEBUGESCAPE+6
   57              FFFF FB42 ALPHAFLAG         EQU DCRTINFO+4
   58              FFFF FB43 GRAPHICSFLAG      EQU ALPHAFLAG+1
```



```
   59              FFFF FB44 GRAPHICSBASE      EQU GRAPHICSFLAG+1
   60              FFFF FB48 INITSR            EQU GRAPHICSBASE+4
   61
   62              0000 4EF9 JMP       EQU $4EF9                       LONG ABSOLUTE JMP INSTRUCTION
   63
   64
   65    00000000                      RORG 0
   66
   67              0000 0000 ASM_INITVECTS     EQU *                   MAIN PROGRAM POWER UP LOCATION
   68
   69    00000000 70FF               MOVEQ   #$FFFFFFFF,D0           HIGHEST POSSIBLE ADDRESS
   70    00000002 2B40 FFFA          MOVE.L  D0,FILELISTPTR(A5)
   71    00000006 31FC 4E75          MOVE.W  #$4E75,DEBUGESCAPE
                  FB38
   72    0000000C 42B8 FB12          CLR.L   LASTLINE
   73    00000010 2B7C 0000          MOVE.L  #ENDISR,ENDISRHOOK(A5)  SET UP ORDINARY RETURN FROM ISR's
                  027A FFBE
   74
   75                                *                               TURN OFF GRAPHICS
   76    00000018 207C 0053          MOVEA.L #G_OFF_MEM,A0           GET MEM ADDRESS
                  8000
   77    0000001E 21C8 FB44          MOVE.L  A0,GRAPHICSBASE         SAVE IT IN GLOBAL AREA
   78    00000022 4238 FB43          CLR.B   GRAPHICSFLAG            MARK IT TURNED OFF
   79
   80    00000026 43FA 013A          LEA     IGNOREBUS,A1            SET BUS ERROR VECTOR
   81    0000002A 21C9 FFFC          MOVE.L  A1,-4
   82    0000002E 31FC 4EF9          MOVE.W  #JMP,-6
                  FFFA
   83    00000034 3010               MOVE    (A0),D0                 TURN OFF GRAPHICS
   84
   85    00000036 11FC 0001          MOVE.B  #1,ALPHAFLAG            MARK ALPHA TURNED ON
                  FB42
   86    0000003C 207C 0051          MOVEA.L #ALPHA_MEM,A0           BLANK ALPHA CRT
                  2000
   87    00000042 303C 07CF          MOVE.W  #2000-1,D0
   88    00000046 7220               MOVEQ   #' ',D1
   89    00000048 30C1       CLEAR_L MOVE.W  D1,(A0)+
   90    0000004A 51C8 FFFC          DBRA    D0,CLEAR_L
   91
   92    0000004E 41F8 FFC4          LEA     $FFFFFFC4,A0            ADDRESS PAST INTERRUPT VECTOR LEVEL 1
   93    00000052 43FA 01B8          LEA     INTERRUPT,A1
   94    00000056 7006               MOVEQ   #6,D0                   HANDLE LEVELS 1 THRU 6
   95    00000058 2109       I_LOOP  MOVE.L  A1,-(A0)                MOVE JMP TO INTERRUPT VECTOR
   96    0000005A 313C 4EF9          MOVE.W  #JMP,-(A0)
   97    0000005E 5340               SUBQ    #1,D0
   98    00000060 66F6               BNE     I_LOOP
   99
  100                                * INIT VECTORS FOR HANDLING <SHIFT PAUSE> AND <CONTROL><SHIFT><PAUSE>
  101    00000062 31FC 4EF9          MOVE.W  #JMP,LEVEL7V
                  FF9A
  102    00000068 21F8 01B0          MOVE.L  $1B0,LEVEL7V+2
                  FF9C
  103    0000006E 31FC 4EF9          MOVE.W  #JMP,KBDRESETV
                  FF34
  104    00000074 21FC 0000          MOVE.L  #RESET_ISR,KBDRESETV+2
                  037A FF36
```

```
PAGE   3 [2.0] 11/4/82 16:14:54  ASSEMBLY OF POWERUP.TEXT

 105    0000007C 31FC 4EF9          MOVE.W  #JMP,CTL_RESETV
                 FB32
 106    00000082 21FC 0000          MOVE.L  #TRYMONITOR,CTL_RESETV+2
                 0398 FB34
 107                         *
 108
 109    0000008A 41FA 026C          LEA     ESC,A0                  MOVE ESCAPE LINKAGE TO HIGH MEMORY
 110    0000008E 31FC 4EF9          MOVE.W  #JMP,ESCAPE
                 FB16
 111    00000094 21C8 FB18          MOVE.L  A0,ESCAPE+2
 112
 113    00000098 2038 FF96          MOVE.L  TRAPOVECTOR+2,D0        GET DEFAULT ENTRY POINT FOR TRAP #0
 114    0000009C C0BC 00FF          AND.L   #$00FFFFFF,D0           TEST FOR MONITOR
                 FFFF
 115    000000A2 B0BC 0088          CMP.L   #$00880000,D0
                 0000
 116    000000A8 6C0E               BGE.S   L0
 117    000000AA 31FC 4EF9          MOVE.W  #JMP,TRAPOVECTOR        SET UP EXCEPTION VECTOR FOR
                 FF94
 118    000000B0 41FA 02BE          LEA     P_BREAK,A0              PASCAL LINE HEADERS
 119    000000B4 21C8 FF96          MOVE.L  A0,TRAPOVECTOR+2
 120
 121                         *==========
 122
 123    000000B8 4CBA 0006 L0       MOVEM.W ESCAPE_PROTO,D1-D2      GET 6 BYTE MOVEQ #ESCCODE, JMP ESCAPE
                 02BA
 124    000000BE 41F8 0000          LEA     0,A0                    START FROM TOP OF RAM
 125    000000C2 303C 4EF9          MOVE.W  #JMP,D0                 $4EF9   (LONG ABS JUMP)
 126
 127    000000C6 43FA 0218          LEA     BUS_ERR,A1              SET UP BUS ERROR TRAP
 128    000000CA 2109               MOVE.L  A1,-(A0)                ADDRESS IN LAST 4 BYTES
 129    000000CC 3100               MOVE.W  D0,-(A0)                LONG JUMP IN FIRST 2 BYTES
 130
 131    000000CE 43FA 0204          LEA     ADD_ERR,A1              SET UP ADDRESS ERROR TRAP
 132    000000D2 2109               MOVE.L  A1,-(A0)                ADDRESS IN LAST 4 BYTES
 133    000000D4 3100               MOVE.W  D0,-(A0)                LONG JUMP IN FIRST 2 BYTES
 134
 135    000000D6 76F3               MOVEQ   #-13,D3                 ILLEGAL INSTRUCTION, ESCAPE(-13)
 136    000000D8 48A0 7000          MOVEM.W D1-D3,-(A0)             MOVE 'moveq, short jump esc'
 137
 138    000000DC 76FB               MOVEQ   #-5,D3                  SIMILAR FOR DIVIDE BY ZERO
 139    000000DE 48A0 7000          MOVEM.W D1-D3,-(A0)
 140
 141    000000E2 76F8               MOVEQ   #-8,D3                  SIMILAR FOR CHK EXCEPTION
 142    000000E4 48A0 7000          MOVEM.W D1-D3,-(A0)
 143
 144    000000E8 76FC               MOVEQ   #-4,D3                  SIMILAR FOR TRAPV
 145    000000EA 48A0 7000          MOVEM.W D1-D3,-(A0)
 146
 147    000000EE 76F2               MOVEQ   #-14,D3                 SIMILAR FOR PRIVILEGE VIOLATION
 148    000000F0 48A0 7000          MOVEM.W D1-D3,-(A0)
 149
 150    000000F4 5D88               SUBQ.L  #6,A0                   SKIP TRACE
 151
 152    000000F6 76F3               MOVEQ   #-13,D3                 ILLEGAL INSTRUCTION (OPS A,F), ESCAPE(-13)
 153    000000F8 48A0 7000          MOVEM.W D1-D3,-(A0)             MOVE 'moveq, short jump esc'
 154    000000FC 48A0 7000          MOVEM.W D1-D3,-(A0)             MOVE 'moveq, short jump esc'
```

```
PAGE   4 [2.0] 11/4/82 16:14:54  ASSEMBLY OF POWERUP.TEXT

 155
 156    00000100 41F8 FF94          LEA     $FFFFFF94,A0            CONTINUE WITH TRAP VECTORS
 157
 158    00000104 43FA 00DE          LEA     LINKA6,A1               TRAP 1, LINK A6 EMULATOR
 159    00000108 2109               MOVE.L  A1,-(A0)
 160    0000010A 3100               MOVE.W  D0,-(A0)
 161
 162    0000010C 76FE               MOVEQ   #-2,D3                  TRAP 2, STACK OVERFLOW
 163    0000010E 48A0 7000          MOVEM.W D1-D3,-(A0)
 164
 165    00000112 76F6               MOVEQ   #-10,D3                 TRAP 3, I/O RESULT NOT ZERO
 166    00000114 48A0 7000          MOVEM.W D1-D3,-(A0)
 167
 168    00000118 76FC               MOVEQ   #-4,D3                  TRAP 4, INTEGER OVERFLOW
 169    0000011A 48A0 7000          MOVEM.W D1-D3,-(A0)
 170
 171    0000011E 76FB               MOVEQ   #-5,D3                  TRAP 5, INTEGER DIVIDE BY ZERO
 172    00000120 48A0 7000          MOVEM.W D1-D3,-(A0)
 173
 174    00000124 76F7               MOVEQ   #-9,D3                  TRAP 6, CASE STATEMENT ERROR
 175    00000126 48A0 7000          MOVEM.W D1-D3,-(A0)
 176
 177    0000012A 76F8               MOVEQ   #-8,D3                  TRAP 7, VALUE RANGE ERROR
 178    0000012C 48A0 7000          MOVEM.W D1-D3,-(A0)
 179
 180    00000130 76FD               MOVEQ   #-3,D3                  TRAP 8, NIL POINTER REFERENCE
 181    00000132 48A0 7000          MOVEM.W D1-D3,-(A0)
 182
 183    00000136 43FA 006C          LEA     NLGOTO,A1               TRAP 9, NON LOCAL GOTO
 184    0000013A 2109               MOVE.L  A1,-(A0)
 185    0000013C 3100               MOVE.W  D0,-(A0)
 186
 187    0000013E 43FA 01D0          LEA     ESCN,A1                 TRAP 10, ESCAPE N
 188    00000142 2109               MOVE.L  A1,-(A0)
 189    00000144 3100               MOVE.W  D0,-(A0)
 190
 191    00000146 43FA 02C0          LEA     SUPERCALL,A1            TRAP 11, SUPERVISOR CALL
 192    0000014A 2109               MOVE.L  A1,-(A0)
 193    0000014C 3100               MOVE.W  D0,-(A0)
 194
 195    0000014E 76EB               MOVEQ   #-21,D3                 TRAP 12, UNASSIGNED
 196    00000150 48A0 7000          MOVEM.W D1-D3,-(A0)
 197
 198    00000154 76EB               MOVEQ   #-21,D3                 TRAP 13, UNASSIGNED
 199    00000156 48A0 7000          MOVEM.W D1-D3,-(A0)
 200
 201    0000015A 76EB               MOVEQ   #-21,D3                 TRAP 14, UNASSIGNED
 202    0000015C 48A0 7000          MOVEM.W D1-D3,-(A0)
 203
 204    00000160 4E75               RTS                             END OF VECTOR SET UP, POWER UP
 205
 206    00000162 DFFC 0000 IGNOREBUS ADDA.L  #8,SP                  THROW AWAY STACK INFO FROM BUS ERROR
                 0008
 207    00000168 4E73               RTE                             RETURN TO POINT OF ERROR
 208
 209                         *=================================================================
 210             0000 016A ASM_CLOSEFILES EQU *
```

```
PAGE   5 [2.0] 11/4/82 16:14:54  ASSEMBLY OF POWERUP.TEXT

 211    0000016A 206F 0004          MOVEA.L 4(SP),A0          EVENTUAL SP
 212    0000016E 226D FFFA          MOVEA.L FILELISTPTR(A5),A1
 213    00000172 B3C8               CMPA.L  A0,A1
 214    00000174 642A               BCC.S   ALLDONE
 215    00000176 B3CF               CMPA.L  SP,A1
 216    00000178 6526               BCS.S   ALLDONE
 217    0000017A 2B69 0004          MOVE.L  4(A1),FILELISTPTR(A5)
                 FFFA
 218    00000180 42A9 0004          CLR.L   4(A1)             FLAG UNINITIALIZED
 219    00000184 2F2D FFEA          MOVE.L  IORESULT(A5),-(SP)
 220    00000188 3F2D FFFE          MOVE.W  ESCAPECODE(A5),-(SP)
 221    0000018C 4851               PEA     (A1)
 222    0000018E 4267               CLR.W   -(SP)             NORMAL CLOSE
 223    00000190 4EB9 0000          JSR     FS_FCLOSE
                 0000
 224    00000196 3B5F FFFE          MOVE.W  (SP)+,ESCAPECODE(A5)
 225    0000019A 2B5F FFEA          MOVE.L  (SP)+,IORESULT(A5)
 226    0000019E 60CA               BRA.S   ASM_CLOSEFILES
 227    000001A0 2E9F       ALLDONE MOVE.L  (SP)+,(SP)
 228    000001A2 4E75               RTS
 229
 230    000001A4 301F       NLGOTO  MOVE.W  (SP)+,D0          SAVE STATUS REG
 231    000001A6 205F               MOVEA.L (SP)+,A0
 232    000001A8 46C0               MOVE    D0,SR             RESTORE USER MODE
 233    000001AA 3218               MOVE.W  (A0)+,D1          STATIC DELTA
 234    000001AC 6C04               BGE.S   NLGOTO1
 235    000001AE 2C78 FB22          MOVEA.L INITSTACK,A6      DEST IS MAIN PROG
 236    000001B2 6F08       NLGOTO1 BLE.S   NLGOTO3
 237    000001B4 2C6E 0008 NLGOTO2 MOVEA.L 8(A6),A6
 238    000001B8 5341               SUBQ.W  #1,D1
 239    000001BA 6EF8               BGT.S   NLGOTO2
 240    000001BC 2210       NLGOTO3 MOVE.L  (A0),D1           DESTINATION DELTA
 241    000001BE 4870 1800          PEA     0(A0,D1.L)        COMPUTE RETURN ADDRESS
 242    000001C2 3430 1802          MOVE.W  2(A0,D1.L),D2     SP DELTA FROM A6
 243    000001C6 43F6 2000          LEA     0(A6,D2.W),A1     EVENTUAL SP
 244    000001CA B3ED FFF6 NLGOTO4 CMPA.L  RECOVERBLOCK(A5),A1 POP OFF TRY RECOVER BLOCKS
 245    000001CE 630C               BLS.S   NLGOTO5           ABOVE THE EVENTUAL SP
 246    000001D0 246D FFF6          MOVEA.L RECOVERBLOCK(A5),A2
 247    000001D4 2B6A 0008          MOVE.L  8(A2),RECOVERBLOCK(A5)
                 FFF6
 248    000001DA 60EE               BRA.S   NLGOTO4
 249    000001DC 2F09       NLGOTO5 MOVE.L  A1,-(SP)
 250    000001DE 4EBA FF8A          JSR     ASM_CLOSEFILES    CLOSE FILES BEING POPPED OFF
 251    000001E2 4E75               RTS
 252
 253    000001E4 301F       LINKA6  MOVE    (SP)+,D0          STATUS
 254    000001E6 205F               MOVEA.L (SP)+,A0          PC
 255    000001E8 46C0               MOVE    D0,SR             RETURN TO ORIGINAL MODE
 256    000001EA 2F0E               MOVE.L  A6,-(SP)          EMULATE LINK A6
 257    000001EC 2C4F               MOVEA.L SP,A6
 258    000001EE DED8               ADDA.W  (A0)+,SP
 259
 260    000001F0 0800 000D          BTST    #13,D0            WHICH MODE?
 261    000001F4 6706               BEQ.S   UMODE
 262
 263    000001F6 45ED 7FFE SMODE    LEA     32766(A5),A2
 264    000001FA 6004               BRA.S   SCHECK
```

```
PAGE   6 [2.0] 11/4/82 16:14:54  ASSEMBLY OF POWERUP.TEXT

 265    000001FC 246D FFF2 UMODE   MOVEA.L HEAPPOINTER(A5),A2
 266
 267    00000200 43EF 0000 SCHECK  LEA     -STACKFUDGE(SP),A1
 268
 269    00000204 B3CA              CMPA.L  A2,A1  CHECK STACK OVERFLOW
 270    00000206 6302              BLS.S   STACKOV
 271    00000208 4ED0              JMP     (A0)
 272
 273    0000020A 4E42       STACKOV TRAP    #2                SIGNAL STACK OVERFLOW
 274
```

```
                        ********************************************************************************
                        *                        INTERRUPT POLLING ROUTINE: LEVELS 1-6                 *
                        ********************************************************************************

                        *
                        * ISRIB STRUCTURE
                        *
               0000 0000 INTREGADDR      EQU     0       (LONG)          CHARPTR
               0000 0004 INTREGMASK      EQU     4       (BYTE)          BYTE
               0000 0005 INTREGVALUE     EQU     5       (BYTE)          BYTE
               0000 0006 CHAINFLAG       EQU     6       (WORD)          BOOLEAN IN MSB; OTHER BITS RESERVED
               0000 0008 PROC.ADDR       EQU     8       (LONG)          PROCEDURE ADDRESS
               0000 000C PROC.LINK       EQU     12      (LONG)          PROCEDURE STATIC LINK
               0000 0010 LINK            EQU     16      (LONG)          LINK TO NEXT ISRIB

                        *
                        *  INTERRUPT POLLING ROUTINE: LEVELS 1-6
                        *
               0000 020C INTERRUPT       EQU *

0000020C 48E7 FFFE              MOVEM.L D0-D7/A0-A6,-(SP)       SAVE CONTEXT, 64 BYTES
00000210 2A78 FB2E              MOVEA.L G_DOLLAR,A5             SET UP THE PASCAL GLOBAL ENVIRONMENT
00000214 2F2D FFEA              MOVE.L  IORESULT(A5),-(SP)      IORESULT IS PART OF THE CONTEXT
00000218 3F2D FFFE              MOVE.W  ESCAPECODE(A5),-(SP)    THE ESCAPE CODE IS PART OF THE CONTEXT

0000021C 2F2D FFF6              MOVE.L  RECOVERBLOCK(A5),-(SP)  SET UP A TRY/RECOVER BLOCK
00000220 487A 0088              PEA     RECOVER
00000224 2B4F FFF6              MOVE.L  SP,RECOVERBLOCK(A5)

00000228 40C0                   MOVE    SR,D0                   FETCH THE CURRENT INTERRUPT LEVEL
0000022A EC48                   LSR     #8-2,D0                 POSITION IN THE LOWER BYTE, MULTIPLIED BY 4
0000022C C07C 001C              AND.W   #$1C,D0                 MASK OUT THE OTHER BITS
00000230 41ED FFC2              LEA     INTERRUPTTABLE(A5),A0
00000234 2070 00FC              MOVEA.L -4(A0,D0),A0            POINT TO THE FIRST ISRIB FOR THIS LEVEL

00000238 2008           PLOOP   MOVE.L  A0,D0                   SET THE CONDITION CODES
0000023A 6722                   BEQ.S   NOISR                   BRANCH IF THE POINTER IS NILL
0000023C 2248                   MOVEA.L A0,A1                   USE A1 FOR SCANNING THE ISRIB
0000023E 2459                   MOVEA.L (A1)+,A2                INTERRUPT REGISTER ADDRESS
00000240 1012                   MOVE.B  (A2),D0                 INTERRUPT REGISTER CONTENTS
00000242 C019                   AND.B   (A1)+,D0                INTERRUPT REGISTER MASK
00000244 B019                   CMP.B   (A1)+,D0                THIS SOURCE REQUESTING AN INTERRUPT?
00000246 6610                   BNE.S   NEXT                    BRANCH IF NOT

00000248 4259                   CLR     (A1)+                   CLEAR THE CHAIN FLAG
0000024A 2F08                   MOVE.L  A0,-(SP)                SAVE THE POINTER TO THIS ISRIB
0000024C 2F08                   MOVE.L  A0,-(SP)                ALSO, THE ISR GETS IT AS A PARAMETER

0000024E 6130                   BSR.S   CALLPROC                CALL THE ISR

00000250 205F                   MOVEA.L (SP)+,A0                RESTORE THE POINTER TO THIS ISRIB
00000252 4A68 0006              TST     CHAINFLAG(A0)           DID THIS ISR CHOOSE TO SERVICE THE INTERRUPT?
00000256 670E                   BEQ.S   RESTORE                 BRANCH IF SO; OTHERWISE...

00000258 2068 0010      NEXT    MOVEA.L LINK(A0),A0             POINT TO THE NEXT ISRIB IN THE LINKED LIST
0000025C 60DA                   BRA     PLOOP                   AND POLL IT's associated interrupt bit
```



```
0000025E 2F08           NOISR   MOVE.L  A0,-(SP)                 NILL POINTER
00000260 4EB9 0000              JSR     INITUNITS_NOISR
                0000

00000266 588F           RESTORE ADDQ.L  #4,SP                   POP OFF THE RECOVER BLOCK ADDRESS
00000268 2B5F FFF6      RECOV_1 MOVE.L  (SP)+,RECOVERBLOCK(A5)  RESTORE THE ORIGINAL RECOVER BLOCK
0000026C 3B5F FFFE              MOVE.W  (SP)+,ESCAPECODE(A5)    RESTORE THE ORIGINAL ESCAPE CODE
00000270 2B5F FFEA              MOVE.L  (SP)+,IORESULT(A5)      RESTORE IORESULT
00000274 206D FFBE              MOVEA.L ENDISRHOOK(A5),A0       CALL ISR HOOK TO ALLOW MULTI-TASKING
00000278 4ED0                   JMP     (A0)

0000027A 4CDF 7FFF      ENDISR  MOVEM.L (SP)+,D0-D7/A0-A6       RESTORE THE ORIGINAL CONTEXT
0000027E 4E73                   RTE                             END OF INTERRUPT SERVICE

                0000 0280 CALLPROC EQU *                        PROCEDURE CALL
00000280 2059                   MOVEA.L (A1)+,A0                PROC ADDRESS
00000282 2019                   MOVE.L  (A1)+,D0                STATIC LINK
00000284 6706                   BEQ.S   CALLIT                  SKIP IF THERE IS NO STATIC LINK
00000286 225F                     MOVEA.L (SP)+,A1              SHUFFLE RETURN ADDRESS
00000288 2F00                     MOVE.L  D0,-(SP)              PUSH STATIC LINK ON THE STACK
0000028A 2F09                     MOVE.L  A1,-(SP)
0000028C 4ED0           CALLIT  JMP     (A0)                    CALL THE PROCEDURE


                        *
                        * RECOVER BLOCK ROUTINES
                        *
0000028E 31EF 0046      RECOV_2 MOVE.W  70(SP),SRTEMP           SAVE THE STATUS REGISTER
                FB20
00000294 3F7C 2100              MOVE.W  #$2100,70(SP)           DUMMY STATUS
                0046
0000029A 2038 FB1C              MOVE.L  PCTEMP,D0               SAVE THE TARGET ADDRESS
0000029E 21EF 0048              MOVE.L  72(SP),PCTEMP           SWITCH PCTEMP
                FB1C
000002A4 2F40 0048              MOVE.L  D0,72(SP)
000002A8 60BE                   BRA.S   RECOV_1

000002AA 0C6D FFEA      RECOVER CMPI.W  #-22,ESCAPECODE(A5)     TEST FOR DEBUGGER CALL
                FFFE
000002B0 67DC                   BEQ     RECOV_2
000002B2 0C6D FFEC              CMPI.W  #-20,ESCAPECODE(A5)     TEST FOR STOP KEY
                FFFE
000002B8 66AE                   BNE     RECOV_1

                        *       STOP KEY PRESSED                SIMULATE AN ESCAPE(-20)
000002BA 2B5F FFF6              MOVE.L  (SP)+,RECOVERBLOCK(A5)  RESTORE THE ORIGINAL RECOVER BLOCK
000002BE 548F                   ADDQ.L  #2,SP                   GET RID OF SAVED ESCAPECODE
000002C0 2B5F FFEA              MOVE.L  (SP)+,IORESULT(A5)      RESTORE IORESULT
000002C4 41FA 000C              LEA     STOP,A0
000002C8 2F48 003E              MOVE.L  A0,62(SP)               REDIRECT INTERRUPTED PROGRAM TO STOP
000002CC 206D FFBE              MOVEA.L ENDISRHOOK(A5),A0       CALL ISR HOOK TO ALLOW MULTI-TASKING
000002D0 4ED0                   JMP     (A0)

000002D2 4E4A           STOP    TRAP    #10

000002D4 2A78 FB2E      ADD_ERR MOVEA.L G_DOLLAR,A5             FIX A5
```

```
 384    000002D8 3B7C FFF5          MOVE.W  #-11,ESCAPECODE(A5)     SET ESCAPE CODE
                 FFFE
 385    000002DE 600A               BRA.S   BE_INFO
 386    000002E0 2A78 FB2E BUS_ERR  MOVEA.L G_DOLLAR,A5             FIX A5
 387    000002E4 3B7C FFF4          MOVE.W  #-12,ESCAPECODE(A5)     SET ESCAPE CODE
                 FFFE
 388    000002EA 31DF FB00 BE_INFO  MOVE.W  (SP)+,BUS_ERR_INFO      SAVE RELEVANT DIAGNOSTIC INFO
 389    000002EE 21DF FB02          MOVE.L  (SP)+,BUS_ERR_ADDR
 390    000002F2 31DF FB06          MOVE.W  (SP)+,BUS_ERR_INSTR
 391    000002F6 6018               BRA.S   ESCN
 392                        *                                       THE REST IS LIKE OTHER EXCEPTIONS
 393
 394    000002F8 2F0D       ESC     MOVE.L  A5,-(SP)                SAVE A5
 395    000002FA 2A6F 0004          MOVEA.L 4(SP),A5                GET CODE POINTER
 396    000002FE 3F55 0006          MOVE.W  (A5),6(SP)              SAVE CODE
 397    00000302 2A78 FB2E          MOVEA.L G_DOLLAR,A5
 398    00000306 3B6F 0006          MOVE.W  6(SP),ESCAPECODE(A5)    SET ERROR CODE
                 FFFE
 399    0000030C 2A5F               MOVEA.L (SP)+,A5                RESTORE A5
 400    0000030E 588F               ADDQ.L  #4,SP                   POP SCRATCH SPACE
 401
 402
 403    00000310 31DF FB08 ESCN     MOVE.W  (SP)+,EXCP_STATUS       SAVE EXCEPTION DIAGNOSTIC INFO
 404    00000314 21DF FB0A          MOVE.L  (SP)+,EXCP_PC
 405    00000318 2F08               MOVE.L  A0,-(SP)                SAVE A0
 406    0000031A 21FC FFFF          MOVE.L  #-1,EXCP_LINE           TRY TO GET LINE #
                 FFFF FB0E
 407    00000322 2078 FB12          MOVEA.L LASTLINE,A0
 408    00000326 0C58 4E40          CMPI.W  #$4E40,(A0)+            TRAP 0 ?
 409    0000032A 6608               BNE.S   NO_LINE
 410    0000032C 4278 FB0E          CLR.W   EXCP_LINE
 411    00000330 31D0 FB10          MOVE.W  (A0),EXCP_LINE+2        FOUND LINE #
 412    00000334 205F       NO_LINE MOVEA.L (SP)+,A0                RESTORE A0
 413    00000336 4EB8 FB38          JSR     DEBUGESCAPE             DEBUGGER HOOK
 414    0000033A 4E4B               TRAP    #11
 415    0000033C 544F               ADDQ    #2,SP
 416    0000033E 2A78 FB2E          MOVEA.L G_DOLLAR,A5             FIX A5
 417
 418                        * TRY TO DETECT CASES OF INTERRUPTED SUPERVISOR CALLS DURING USER PROGRAM
 419    00000342 206D FFF6          MOVEA.L RECOVERBLOCK(A5),A0
 420    00000346 B1CF               CMPA.L  SP,A0
 421    00000348 6C08               BGE.S   TEST_2
 422    0000034A 08B8 0005          BCLR    #5,EXCP_STATUS          MOVE BACK TO USER MODE BECAUSE USER
                 FB08
 423    00000350 600C               BRA.S   NORMAL_RECOVER            STACK IS BELOW SUPERVISOR STACK
 424    00000352 B1F8 FB2A TEST_2   CMPA.L  INITRECOVER,A0
 425    00000356 6606               BNE.S   NORMAL_RECOVER
 426    00000358 31F8 FB48          MOVE    INITSR,EXCP_STATUS      INITSR WAS SAVED WHEN INITRECOVER WAS
                 FB08
 427
 428              0000 035E NORMAL_RECOVER  EQU *
 429    0000035E 46F8 FB08          MOVE    EXCP_STATUS,SR          RETURN TO USER MODE
 430    00000362 2F2D FFF6          MOVE.L  RECOVERBLOCK(A5),-(SP)  GO CLOSE ALL FILES ABOVE NEW (SP)
 431    00000366 6100 FE02          BSR     ASM_CLOSEFILES
 432    0000036A 2E6D FFF6          MOVEA.L RECOVERBLOCK(A5),SP     CUT BACK USER's stack
 433    0000036E 4E75               RTS
 434
```



```
 435    00000370 54AF 0002 P_BREAK  ADDQ.L #2,2(SP)                 BUMP RETURN ADDRESS TO SKIP LINE NUMBER
 436    00000374 4E73               RTE                             RETURN FROM TRAP #0
 437
 438    00000376 4EB8 FB16 ESCAPE_PROTO    JSR     ESCAPE
```

```
                        *
                        * HANDLING LEVEL 7 INTS
                        *
            FF88 0024 MONITOR     EQU  $FF880024
            0042 8003 KBDSTATUS   EQU  $00428003
            0042 8001 KBDDATA     EQU  $00428001
            0042 8003 KBDCOMMAND      EQU  $00428003
                        *
0000037A 48E7 FFFE RESET_ISR   MOVEM.L D0-D7/A0-A6,-(SP)   SAVE REGISTERS
0000037E 13FC 00B2             MOVE.B #$B2,KBDSTATUS
            0042 8003
00000386 614E                  BSR.S  GETCTRL
00000388 614C                  BSR.S  GETCTRL
0000038A 0801 0001             BTST   #1,D1
0000038E 6616                  BNE.S  RESETX
00000390 4CDF 7FFF             MOVEM.L     (SP)+,D0-D7/A0-A6       RESTORE REGISTERS
00000394 4EF8 FB32             JMP    CTL_RESETV       WATCH IF OUT OF RANGE

00000398 4879 FF88 TRYMONITOR  PEA    MONITOR
            0024
0000039E 4EB8 01A0             JSR    RTN_TO_MONITOR
000003A2 48E7 FFFE             MOVEM.L D0-D7/A0-A6,-(SP)   SAVE REGISTERS
                        *
            0000 03A6 RESETX       EQU *
000003A6 4878 0007             PEA         7                          DEBUGGER(7,0,0)
000003AA 42A7                  CLR.L       -(SP)
000003AC 42A7                  CLR.L       -(SP)
000003AE 2A78 FB2E             MOVEA.L     G_DOLLAR,A5
000003B2 43ED FEEC             LEA         DEBUGGER(A5),A1
000003B6 6100 FEC8             BSR         CALLPROC
000003BA 4CDF 7FFF             MOVEM.L     (SP)+,D0-D7/A0-A6       RESTORE REGISTERS
000003BE 4EB8 FB38             JSR         DEBUGESCAPE             GIVE DEBUGGER ANOTHER SHOT

000003C2 4E70                  RESET                               NO DEBUGGER, GIVE UP, CLEAR I/O
000003C4 7008                  MOVEQ       #8,D0                   WAIT 1 SECOND
000003C6 51C9 FFFE LP1         DBRA        D1,LP1
000003CA 51C8 FFFA             DBRA        D0,LP1
000003CE 4278 FDC2             CLR         -574                    THEN REBOOT FROM SCRATCH
000003D2 4EF8 01C0             JMP         448

                        *
000003D6 6122          GETCTRL     BSR.S  STALL
000003D8 13FC 0005             MOVE.B #5,KBDCOMMAND
            0042 8003
000003E0 6118          GA          BSR.S  STALL
000003E2 0800 0000             BTST   #0,D0
000003E6 67F8                  BEQ    GA
000003E8 1239 0042             MOVE.B KBDDATA,D1
            8001
000003EE C07C 00F0             AND    #$F0,D0
000003F2 B07C 0040             CMP    #$40,D0
000003F6 66E8                  BNE    GA
000003F8 4E75                  RTS
000003FA 1039 0042 STALL       MOVE.B KBDSTATUS,D0
            8003
00000400 0800 0001             BTST   #1,D0
00000404 66F4                  BNE    STALL
```



```
00000406 4E75                  RTS


            0000 0408 SUPERCALL EQU *                  TRAP #11, GETS INTO SUPERVISOR MODE, SAVES SR
00000408 2F2F 0002             MOVE.L  2(SP),-(SP)     COPY RETURN ADDRESS
0000040C 3F6F 0004             MOVE    4(SP),8(SP)     MOVE STATUS REGISTER UP
            0008
00000412 2E9F                  MOVE.L  (SP)+,(SP)      MOVE RETURN ADDRESS UP
00000414 4E75                  RTS                     RETURN TO CALLER IN SUPERVISOR MODE

                        *
                        *
            0000 0416 ASM_INTLEVEL    EQU *                    FUNCTION TO RETURN INTERRUPT LEVEL
00000416 40C0                  MOVE    SR,D0                    GET STATUS REGISTER
00000418 C0BC 0000             AND.L   #$00000700,D0            EXTRACT LEVEL
            0700
0000041E E048                  LSR     #8,D0                    MOVE IT OVER
00000420 2F40 0004             MOVE.L  D0,4(SP)                 RETURN INTEGER RESULT
00000424 4E75                  RTS
                        *
            0000 0426 ASM_SETINTLEVEL   EQU *                   PROCEDURE TO SET INTERRUPT LEVEL
00000426 205F                  MOVEA.L (SP)+,A0                 RETURN ADDRESS
00000428 201F                  MOVE.L  (SP)+,D0                 INTEGER PARAMETER
0000042A C07C 0007             AND     #7,D0                    TAKE MOD 8 FOR SAFETY
0000042E E148                  LSL     #8,D0                    MOVE IT OVER
00000430 4E4B                  TRAP    #11                      MOVE INTO SUPERVISOR MODE
00000432 321F                  MOVE    (SP)+,D1                 GET CURRENT STATUS REGISTER
00000434 C27C F8FF             AND     #$F8FF,D1                CLEAR CURRENT LEVEL
00000438 8041                  OR      D1,D0                    COMBINE WITH NEW LEVEL
0000043A 46C0                  MOVE    D0,SR                    RESTORE STATUS
0000043C 4ED0                  JMP     (A0)

0000043E 205F          ASM_IAND MOVEA.L (SP)+,A0
00000440 201F                  MOVE.L  (SP)+,D0
00000442 C09F                  AND.L   (SP)+,D0
00000444 2E80                  MOVE.L  D0,(SP)
00000446 4ED0                  JMP     (A0)

00000448 205F          ASM_IOR  MOVEA.L (SP)+,A0
0000044A 201F                  MOVE.L  (SP)+,D0
0000044C 809F                  OR.L    (SP)+,D0
0000044E 2E80                  MOVE.L  D0,(SP)
00000450 4ED0                  JMP     (A0)


                        *       TRAPS:
                        *       ADDRESS          TRAP    ESCAPECODE      USUAL MEANING
                        *       $FFFFFF3A        15      (NONE)          DEBUGGER
                        *       $FFFFFF40        14      -21             UNASSIGNED
                        *       $FFFFFF46        13      -21             UNASSIGNED
                        *       $FFFFFF4C        12      -21             UNASSIGNED
                        *       $FFFFFF52        11      -21             UNASSIGNED
                        *       $FFFFFF58        10      "N"             ESCAPE N
                        *       $FFFFFF5E        9       (NONE)          NON LOCAL GOTO
                        *       $FFFFFF64        8        -3             DEREFERENCE NIL POINTER
```

```
 547                        *       $FFFFFF6A       7        -8             VALUE RANGE ERROR
 548                        *       $FFFFFF70       6        -9             CASE STATEMENT ERROR
 549                        *       $FFFFFF76       5        -5             DIVIDE BY ZERO
 550                        *       $FFFFFF7C       4        -4             INTEGER OVERFLOW
 551                        *       $FFFFFF82       3        -3             IORESULT <> 0
 552                        *       $FFFFFF88       2        -2             STACK OVERFLOW
 553                        *       $FFFFFF8E       1        (-2 IF ANY)    LINK A6 EMULATOR WITH STACK CHECK
 554                        *       $FFFFFF94       0        (NONE)         PASCAL LINE BREAKPOINT
 555
 556
 557                        *       SYSTEM EXCEPTIONS:
 558
 559                        *       ADDRESS                  ESCAPECODE     USUAL MEANING
 560
 561                        *       $FFFFFFC4                -13            1111 OPCODE
 562                        *       $FFFFFFCA                -13            1010 OPCODE
 563                        *       $FFFFFFD0                (NONE)         TRACE
 564                        *       $FFFFFFD6                -14            PRIVILEGE VIOLATION
 565                        *       $FFFFFFDC                 -4            INTEGER OVERFLOW (TRAPV)
 566                        *       $FFFFFFE2                 -8            CHK INSTRUCTION
 567                        *       $FFFFFFE8                 -5            DIVIDE BY ZERO (HARDWARE)
 568                        *       $FFFFFFEE                -13            ILLEGAL INSTRUCTION
 569                        *       $FFFFFFF4                -11            ADDRESS ERROR
 570                        *       $FFFFFFFA                -12            BUS ERROR
 571
 572                                NOSYMS
 573                                END .
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

RANDOM

Purpose

RANDOM provides pseudo-random number sequences.

Usage

RANDOM is part of LIBRARY and is accessible by IMPORTing RND.

Requirements

The user must declare and initialize his own seed, which should be a positive 32-bit integer.

```
PASS 1 COMPLETE. ERRORS: 0
    1                                   mname   rnd
    2
    3                                   src     module rnd;
    4                                   src     import sysglobals;
    5                                   src     export
    6                                   src       procedure random(var seed : integer);
    7                                   src       function rand( var seed : integer;
    8                                   src                      range : shortint) : shortint;
    9                                   src     end;
   10
   11                                   def     rnd_rnd
   12                                   def     rnd_random
   13                                   def     rnd_rand
   14                                   def     random
   15                                   def     rand
   16
   17                           ********************************************************************************
   18                           *
   19                           *       Procedure RANDOM(VAR SEED: INTEGER)
   20                           *
   21                           *       Description:
   22                           *               Generate a pseudo-random number with the formula
   23                           *               Xn <- (16807 * Xn-1) MOD (2^31 - 1), where Xn-1 is the
   24                           *               previous random number. A shortcut computation is:
   25                           *                   C <- 16807 * Xn-1.
   26                           *                   Xn <- C MOD 2^31 + C DIV 2^31.
   27                           *                   If Xn > 2^31 - 1, then Xn <- Xn - (2^31 - 1)
   28                           *
   29                           *       Parameters:
   30                           *               rndseed - the previous random number
   31                           *
   32                           *       Error conditions:
   33                           *               There are none.
   34                           ************************************************
   35    00000000 4E75          rnd_rnd rts
   36             0000 0002     rnd_random equ *
   37    00000002 205F          random  movea.l (sp)+,a0        return address
   38    00000004 225F                  movea.l (sp)+,a1        address of seed
   39    00000006 2011                  move.l  (a1),d0         get previous random seed Xn
   40    00000008 2200                  move.l  d0,d1           leave bottom 16 bits in d0
   41    0000000A 4841                  swap    d1              get top 16 bits into d1
   42    0000000C C0FC 41A7             mulu    #16807,d0       get one partial product in d0
   43    00000010 C2FC 41A7             mulu    #16807,d1       high order partial product in d1
   44    00000014 4841                  swap    d1              align middle 16 bits of product in high d1
   45    00000016 D241                  add.w   d1,d1           most of (product div 2^31) is in low d1
   46    00000018 D081                  add.l   d1,d0           compute (product mod 2^31) + (product div 2^31)
   47    0000001A 6402                  bcc.s   rnd1            any carries out of 32nd bit are part of the div
   48    0000001C 5480                  addq.l  #2,d0           (so propagate into appropriate position)
   49    0000001E 6A06          rnd1    bpl.s   rnd2            bit 31 is also part of the div
   50    00000020 90BC 7FFF             sub.l   #$7FFFFFFF,d0         so remove it and add it back to bit 0
                  FFFF
   51    00000026 2280          rnd2    move.l  d0,(a1)
   52    00000028 4ED0                  jmp     (a0)
   53
   54                           ************************************************
   55                           *
   56                           *       Function RAND(VAR SEED: INTEGER;
   57                           *               RANGE: SHORTINT): SHORTINT
```



```
   58                           *
   59                           *       Returns a 16 bit integer which is scaled
   60                           *       to the range 0..RANGE-1
   61                           *       (RANGE is treated as unsigned!)
   62                           ************************************************
   63
   64             0000 002A     rnd_rand equ *
   65    0000002A 245F          rand    movea.l (sp)+,a2        return address
   66    0000002C 341F                  move.w  (sp)+,d2        range parameter
   67    0000002E 61D2                  bsr.s   rnd_random      compute into d0
   68    00000030 E380                  asl.l   #1,d0           normalize
   69    00000032 4840                  swap    d0                   to 16 bits
   70    00000034 C0C2                  mulu    d2,d0           scale to range
   71    00000036 4840                  swap    d0
   72    00000038 3E80                  move.w  d0,(sp)         return result
   73    0000003A 4ED2                  jmp     (a2)
   74
   75                                   end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

*** NO EXTERNAL SYMBOLS ***

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
CCR	STREG	0		00000005
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
RAND	REL	65		0000002A
RANDOM	REL	37		00000002
RND1	REL	49		0000001E
RND2	REL	51		00000026
RND_RAND	REL	64		0000002A
RND_RANDOM	REL	36		00000002
RND_RND	REL	35		00000000
SP	AREG	0		00000007
SR	STREG	0		00000006
USP	STREG	0		00000007

ROMCALL

Purpose

ROMCALL provides an interface to several Boot ROM routines, swapping environments appropriately.

```
PAGE   1 [2.0] 11/4/82 13:13:45  ASSEMBLY OF ROMCALL.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                                   mname   ROMcall
    2
    3                                   def     asm_f_pwr_on,asm_flpyread,asm_flpy_wrt,asm_flpyinit
    4                                   def     asm_flpymread,asm_flpymwrite
    5                                   def     boot_lifhead,boot_findfile
    6                                   def     boot_minit,boot_mfopen,boot_mfclose,boot_mread
    7                                   refa    sysglobals
    8               FFFF FFFE escapecode        equ sysglobals-2
    9               FFFF FFF6 recoverblock      equ sysglobals-10
   10               FFFF FFB8 intvec2           equ -72                 vector for interrupt level 2
   11
   12                         *  hard coded addresses:
   13               0000 0120 flpyread          equ $120        288
   14               0000 0124 flpy_wrt          equ $124        292
   15               0000 0128 fintrupt          equ $128        296
   16               0000 012C flpyinit          equ $12C        300
   17               0000 0130 flpymread         equ $130        304
   18               0000 0134 flpymwrite        equ $134        308
   19               0000 0144 f_pwr_on          equ $144        324
   20               0000 018C lifhead           equ     $18C
   21                         *findfile         equ     $190
   22
   23               0000 4004 minit             equ     $4004
   24               0000 4008 mfopen            equ     $4008
   25               0000 4010 mfclose           equ     $4010
   26               0000 400C mread             equ     $400C
   27
   28               0000 0000 A         equ     a0      address of boot ROM routine
   29               0000 0004 R         equ     a4      return address
   30               0000 0000 F         equ     d0      function result
   31
   32
   33
   34               0000 0000 boot_mfopen       equ  *
   35    00000000 41F8 4008           lea     mfopen,A
   36
   37    00000004 285F        f2_16   movea.l (sp)+,R                 get return address
   38    00000006 4CDF 001E           movem.l (sp)+,d1-d4             get parameters
   39    0000000A 2F0C                move.l  R,-(sp)                 put return address back
   40    0000000C 613C                bsr.s   fixit                   get into boot ROM environment
   41    0000000E 4267                clr.w   -(sp)                   function result
   42    00000010 48E7 7800           movem.l d1-d4,-(sp)             push parameters
   43    00000014 6010                bra.s   rom_fun
   44
   45
   46
   47               0000 0016 boot_minit        equ  *
   48    00000016 41F8 4004           lea     minit,A
   49
   50    0000001A 285F        f2_4    movea.l (sp)+,R                 get return address
   51    0000001C 221F                move.l  (sp)+,d1                get parameters
   52    0000001E 2F0C                move.l  R,-(sp)                 put return address back
   53    00000020 6128                bsr.s   fixit                   get into boot ROM environment
   54    00000022 4267                clr.w   -(sp)                   function result
   55    00000024 2F01                move.l  d1,-(sp)                push parameters
   56
   57    00000026 4E90        rom_fun jsr     (A)                     call boot ROM
   58    00000028 301F                move.w  (sp)+,F                 get function result


PAGE   2 [2.0] 11/4/82 13:13:45  ASSEMBLY OF ROMCALL.TEXT

   59    0000002A 6140                bsr.s   unfix                   return to user environment
   60    0000002C 3F40 0004           move.w  F,4(sp)                 put back function result
   61    00000030 4E75                rts                             return
   62
   63
   64
   65               0000 0032 asm_flpyinit equ *
   66    00000032 41F8 012C           lea     flpyinit,A
   67
   68    00000036 285F        byte_6  movea.l (sp)+,R                 get return address
   69    00000038 4C9F 000E           movem.w (sp)+,d1-d3             get parameters
   70    0000003C 2F0C                move.l  R,-(sp)                 put return address back
   71    0000003E 610A                bsr.s   fixit                   get into boot ROM environment
   72    00000040 48A7 7000           movem.w d1-d3,-(sp)             push parameters
   73
   74    00000044 4E90        rom_jsr jsr     (A)                     call boot ROM
   75    00000046 6124                bsr.s   unfix                   restore user envirnoment
   76    00000048 4E75                rts                             return to user
   77
   78
   79
   80    0000004A 285F        fixit   movea.l (sp)+,R                 fetch return address
   81    0000004C 4E4B                trap    #11                     get into supervisor mode
   82    0000004E 47F8 FFB8           lea     intvec2,a3              save floppy vector
   83    00000052 3F1B                move.w  (a3)+,-(sp)             save jmp
   84    00000054 2F1B                move.l  (a3)+,-(sp)             save address
   85    00000056 273C 0000           move.l  #fintrupt,-(a3)         set up boot ROM floppy vector
                  0128
   86    0000005C 373C 4EF9           move.w  #20217,-(a3)            (jmp)
   87    00000060 4E55 FFFE           link    a5,#-2                  primitive try/recover
   88    00000064 487A 0076           pea     rom_recover
   89    00000068 4857                pea     (sp)
   90    0000006A 4ED4                jmp     (R)                     return to call
   91
   92    0000006C 285F        unfix   movea.l (sp)+,R                 fetch return address
   93    0000006E 4E5D                unlk    a5                      pop recover stuff
   94    00000070 47F8 FFBE           lea     intvec2+6,a3            restore floppy vector
   95    00000074 271F                move.l  (sp)+,-(a3)             address
   96    00000076 371F                move.w  (sp)+,-(a3)             jmp
   97    00000078 46DF                move    (sp)+,sr                restore user mode
   98    0000007A 4ED4                jmp     (R)                     return to call
   99
  100
  101
  102               0000 007C boot_mread        equ  *
  103    0000007C 41F8 400C           lea     mread,A
  104
  105    00000080 285F        byte_14 movea.l (sp)+,R                 get return address
  106    00000082 4C9F 00FE           movem.w (sp)+,d1-d7             get parameters
  107    00000086 2F0C                move.l  R,-(sp)                 put return address back
  108    00000088 61C0                bsr.s   fixit                   get into boot ROM environment
  109    0000008A 48A7 7F00           movem.w d1-d7,-(sp)             push parameters
  110    0000008E 60B4                bra.s   rom_jsr
  111
  112
  113
  114               0000 0090 boot_mfclose    equ *
```

```
00000090 41F8 4010          lea     mfclose,A
00000094 6004               bra.s   no_parm

            0000 0096 asm_f_pwr_on equ *
00000096 41F8 0144          lea     f_pwr_on,A                  boot ROM mini reset

0000009A 61AE       no_parm bsr.s   fixit                       get into boot ROM environment
0000009C 60A6               bra.s   rom_jsr



            0000 009E boot_lifhead equ *
0000009E 41F8 018C          lea     lifhead,A
000000A2 600A               bra.s   byte_8

            0000 00A4 asm_flpy_wrt equ *
000000A4 41F8 0124          lea     flpy_wrt,A                  boot ROM single-sector write
000000A8 6004               bra.s   byte_8

            0000 00AA asm_flpyread equ *
000000AA 41F8 0120          lea     flpyread,A                  boot ROM single-sector read

000000AE 285F       byte_8  movea.l (sp)+,R                     get return address
000000B0 4CDF 0006          movem.l (sp)+,d1-d2                 get parameters
000000B4 2F0C               move.l  R,-(sp)                     put return address back
000000B6 6192               bsr.s   fixit                       get into boot ROM environment
000000B8 48E7 6000          movem.l d1-d2,-(sp)                 push parameters
000000BC 6086               bra.s   rom_jsr



            0000 00BE asm_flpymwrite equ *
000000BE 41F8 0134          lea     flpymwrite,A                boot ROM multisector write
000000C2 6004               bra.s   byte_12

            0000 00C4 asm_flpymread equ *
000000C4 41F8 0130          lea     flpymread,A                 boot ROM multisector read

000000C8 285F       byte_12 movea.l (sp)+,R                     get return address
000000CA 4CDF 000E          movem.l (sp)+,d1-d3                 get parameters
000000CE 2F0C               move.l  R,-(sp)                     put return address back
000000D0 6100 FF78          bsr     fixit                       get into boot ROM environment
000000D4 48E7 7000          movem.l d1-d3,-(sp)                 push parameters
000000D8 6000 FF6A          bra     rom_jsr



            0000 00DC rom_recover equ *                         escape from boot ROM
000000DC 301F               move.w  (sp)+,d0                    get escapecode
000000DE 618C               bsr.s   unfix                       return to user environment
000000E0 3B40 FFFE          move.w  d0,escapecode(a5)           restore escapecode
000000E4 4E4A               trap    #10                         escape(escapecode)



            FFFF FDD2 FUBUFFER EQU     $FFFFFDD2       FLOPPY UNIVERSAL BUFFER (RQ)
```



```
            0000 00E6 BOOT_FINDFILE   EQU *
000000E6 225F               MOVEA.L (SP)+,A1            RETURN ADDRESS
000000E8 241F               MOVE.L  (SP)+,D2            GET DIR START
000000EA 261F               MOVE.L  (SP)+,D3            GET DIR LEN
000000EC 205F               MOVEA.L (SP)+,A0            GET FILE NAME ADDR
000000EE 285F               MOVEA.L (SP)+,A4            GET ADDR OF CAT ENTRY
000000F0 2F09               MOVE.L  A1,-(SP)            RESTORE RETURN ADDRESS
000000F2 45F8 FDD2 NXTDSECTOR LEA   FUBUFFER,A2
000000F6 48E7 30A8          MOVEM.L D2/D3/A0/A2/A4,-(SP)    SAVE REGS
000000FA 2F02               MOVE.L  D2,-(SP)
000000FC 2F0A               MOVE.L  A2,-(SP)            READ THE NEXT
000000FE 61AA               BSR.S   ASM_FLPYREAD        DIRECTORY SECTOR
00000100 4CDF 150C          MOVEM.L (SP)+,D2/D3/A0/A2/A4 RESTORE REGS
00000104 7208               MOVEQ   #8,D1               (8 ENTRIES/SECTOR)
00000106 0C6A FFFF CHECKTYPE CMPI.W #-1,10(A2)          CHECK LOGICAL EOD
            000A
0000010C 6726               BEQ.S   NOTFOUND
0000010E 4A6A 000A          TST.W   10(A2)              IGNORE PURGED FILES
00000112 6712               BEQ.S   NXTDENTRY
00000114 2248               MOVEA.L A0,A1               SAVE NAME POINTER
00000116 264A               MOVEA.L  A2,A3              COPY ENTRY POINTER
00000118 7804               MOVEQ    #4,D4              SET CHARACTER COUNTER
0000011A B749       CHKNAME CMPM.W  (A1)+,(A3)+         MATCH CHARACTERS
0000011C 56CC FFFC          DBNE    D4,CHKNAME
00000120 6604               BNE.S   NXTDENTRY
00000122 288A               MOVE.L  A2,(A4)             STORE THE CAT ENTRY ADDR.
00000124 4E75               RTS

00000126 D4FC 0020 NXTDENTRY ADDA   #32,A2              INCREMENT
0000012A 5381               SUBQ.L  #1,D1               TO NEXT ENTRY
0000012C 6ED8               BGT.S   CHECKTYPE
0000012E 5282               ADDQ.L  #1,D2               INCREMENT
00000130 5383               SUBQ.L  #1,D3               TO NEXT DIRECTORY
00000132 6EBE               BGT.S   NXTDSECTOR          SECTOR
00000134 3B7C FFFF NOTFOUND MOVE.W  #-1,ESCAPECODE(A5)  ESCAPECODE := -1
            FFFE
0000013A 4E4A               TRAP    #10

                            NOSYMS
                            END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

RS

Purpose

RS contains assembly language low-level I/O drivers.

Usage

RS is used for the 98626 interface and the 9816 computer's built-in RS-232 interface.

Requirements

RS_DRV and COMASM

```
PASS 1 COMPLETE. ERRORS: 0
    2                       ******************************************************************************
    3                       *
    4                       *       COPYRIGHT (C) 1982 BY HEWLETT-PACKARD COMPANY
    5                       *
    6                       ******************************************************************************
    7                       *
    8                       *
    9                       *       IOLIB     RS
   10                       *
   11                       *
   12                       ******************************************************************************
   13                       *
   14                       *
   15                       *
   16                       *       Library - IOLIB
   17                       *
   18                       *       Purpose - This set of assembly language code is intended to be used as
   19                       *                 a PASCAL module for I/O drivers for use by the external I/O
   20                       *                 procedures library.
   21                       *
   22                       *
   23                       *       Date    - 8-6-82
   24                       *       Update  - 9-30-82
   25                       *       Release - 2.0  (QA6)
   26                       *
   27                       *
   28                       *       Source  - RS: RS.TEXT, RS_HEAD.TEXT, RS_RW.TEXT, RS_CS.TEXT,
   29                       *                     RS_UTIL.TEXT    (5.25" floppy)
   30                       *       Object  - RS: RS.CODE         (5.25" floppy)
   31                       *
   32                       *
   33                       *       Backup  - BACK2: $            (5.25" floppy)
   34                       *
   35                       *
   36                       ******************************************************************************
   37                       *
   38                       *
   39                       *       RELEASED
   40                       *       VERSION
   41                       *
   42                       *
   43                       ******************************************************************************
```



```
   45                       ******************************************************************************
   46                       *
   47                       *
   48                       *       The following lines are used to tell the LINKER/LOADER what this
   49                       *       module looks like in PASCAL terms.
   50                       *
   51                       *       Note that it is possible to create assembly modules that are functions.
   52                       *       These routines are called through an indirect pointer using the CALL
   53                       *       facility which does NOT permit functions.
   54                       *
   55                       *       This module is called 'RS' ( upper or lower case - doesn't matter )
   56                       *       independent of the file name ( by use of the MNAME pseudo-op ).
   57                       *
   58                       *       All the externally used procedures are called 'RS_@@@@@@@' in
   59                       *       this module.  If you are using assembly to access them use the
   60                       *       'RS_@@@@@@@' name.  If you are using Pascal use the '@@@@@@@'
   61                       *       name.
   62                       *
   63                       ******************************************************************************
   64
   65                               MNAME RS
   66
   67                               SRC MODULE RS;
   68                               SRC IMPORT iodeclarations;
   69                               SRC EXPORT
   70                               SRC
   71                               SRC         PROCEDURE rs_init     ( temp : ANYPTR );
   72                               SRC         PROCEDURE rs_isr      ( temp : PISRIB );
   73                               SRC         PROCEDURE rs_rdb      ( temp : ANYPTR ; VAR x : CHAR );
   74                               SRC         PROCEDURE rs_wtb      ( temp : ANYPTR ;  val  : CHAR );
   75                               SRC         PROCEDURE rs_rdw      ( temp : ANYPTR ; VAR x : io_word);
   76                               SRC         PROCEDURE rs_wtw      ( temp : ANYPTR ;  val  : io_word);
   77                               SRC         PROCEDURE rs_rds      ( temp : ANYPTR ;  reg  : io_word;
   78                               SRC                                                VAR x : io_word);
   79                               SRC         PROCEDURE rs_wtc      ( temp : ANYPTR ;  reg  : io_word;
   80                               SRC                                                 val  : io_word );
   81                               SRC         PROCEDURE rs_tfr      ( temp : ANYPTR ;  bcb  : ANYPTR);
   82                               SRC END; { of extdc }
```

```
PAGE   3 [2.0] 11/4/82 16:25:44 IOLIB RS - RS232 DRIVERS

                        ********************************************************************************
                        *
                        *       SYMBOLS FOR EXPORT AS PROCEDURE NAMES
                        *
                        ********************************************************************************

                                DEF RS_RS

                                DEF RS_RS_INIT
                                DEF RS_RS_ISR
                                DEF RS_RS_RDB
                                DEF RS_RS_WTB
                                DEF RS_RS_RDW
                                DEF RS_RS_WTW
                                DEF RS_RS_RDS
                                DEF RS_RS_WTC
                                DEF RS_RS_TFR

                        ********************************************************************************
                        *
                        *       SYMBOLS FOR IMPORT
                        *
                        ********************************************************************************

                                LMODE   ABORT_IO,LOGEOT
                                REFA    ABORT_IO,LOGEOT


                                INCLUDE IOLIB:COMDCL
```

```
PAGE   4 [2.0] 11/4/82 16:25:44 IOLIB COMDCL - COMMON EQUATES AND DEFINITIONS

                        ********************************************************************************
                        *
                        *       modified:  02/22/82  JPC   added parm to user EOT & ISR proc's
                        *
                        ********************************************************************************
                        *
                        *       HPL CONVENTIONS
                        *
                        *
                        *         Much of this code is taken intact from the 9826 HPL
                        *         language system EIO ROM ( extended I/O ROM ).
                        *
                        *         The Pascal that will be calling this code uses
                        *         the stack for parameter passage.  The HPL code
                        *         uses the Ax and Dx registers for all parameters.
                        *         The Pascal driver entry points on the previous pages
                        *         take care of getting the parameters into the correct
                        *         registers.
                        *
                        *
                        *       GENERAL HPL ENTRY/EXIT CONDITIONS:
                        *
                        *         A1.L = CARD ADDRESS
                        *         A2.L = DRIVER TEMP ADDRESS
                        *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
                        *
                        *
                        *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
                        *
                        *         A3.L = BUFFER CONTROL BLOCK ADDRESS
                        *                In addition to the A1/A2 convention, Pascal will use
                        *                A3 for a pointer to the buffer control block.
                        *                The HPL system kept much of the transfer
                        *                information in the s.c. temps.
                        *
                        *         TIMEOUT(A2) = contains timeout information
                        *                Timeout was a global temp in HPL and a timeout
                        *                generated an error.
                        *                In PASCAL each card has a timeout value stored in
                        *                its temporary area.  A timeout error
                        *                generates an ESCAPE ( which can be trapped ).
                        *
                        *
                        ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *          DRIVER TEMPS TEMPLATE
                        *
                        *            OFFSET FROM A2
                        *
                        *            HPL DECLARATIONS ( MODIFIED )
                        *
                        *
                        ********************************************************************************
0000 0000 ISR_ENTRY     EQU 0       ..19        PASCAL ISR LINK & UNLINK area
0000 0014 USER_ISR      EQU 20                  user ISR:  do NOT change the proc/stat link/parm ordering!!!
0000 0014 H_ISR_PR      EQU 20      ..23            procedure ptr
0000 0018 H_ISR_SL      EQU 24      ..27            static link
0000 001C H_ISR_PM      EQU 28      ..31            parameter
0000 0020 C_ADR         EQU 32      ..35        card address
0000 0024 BUFI_OFF      EQU 36      ..39        buffer pointer offset
0000 0028 BUFO_OFF      EQU 40      ..43        buffer pointer offset
0000 002C EIRB_OFF      EQU 44                  eir byte
0000 002D IO_SC         EQU 45                  select code ( i.e. 7, 22, etc. )
0000 002E TIMEOUT       EQU 46      ..49        timeout value
                        *                           =0 : no timeout
                        *                           #0 : value of timeout
0000 0032 MA_W          EQU 50      ..51        word access to my address
0000 0033 MA            EQU 51                  byte access to my address
0000 0034 AVAIL_OFF     EQU 52      ..??        standard space taken from temps
                        *               52      ..83        normal cards ( 32  bytes )
                        *               52      ..179       98628  card  ( 128 bytes )
```



```
                        ********************************************************************************
                        *
                        *       TRANSFER OFFSETS IN BUFFER CONTROL BLOCK
                        *
                        *           OFFSET FROM A3
                        *
                        *           PASCAL DECLARATION
                        *
                        ********************************************************************************
0000 0000 TTMP_OFF      EQU 0   ..3         pointer to driver temp offset
0000 0005 T_SC_OFF      EQU 5               transfer select code
0000 0007 TACT_OFF      EQU 7               actual transfer mode
0000 0009 TUSR_OFF      EQU 9               transfer mode
                        *                       00   -                  not used
                        *                       01   serial DMA
                        *                       02   serial FHS
                        *                       03   serial FASTEST ( DMA or FHS )
                        *                       04   -                  not used
                        *                       ---------------
                        *                       05   overlp INTR
                        *                       06   overlp DMA
                        *                       07   overlp FHS      ( BURST )
                        *                       08   overlp FASTEST ( DMA or BURST )
                        *                       09   overlp OVERLAP ( DMA or INTR )
0000 000A T_BW_OFF      EQU 10              transfer byte/word indicator
                        *                       0 = byte   / 1 = word
0000 000B TEND_OFF      EQU 11              transfer EOI/END indicator
                        *                       0 = no eoi / 1 = eoi sent or searched for
0000 000D TDIR_OFF      EQU 13              transfer direction
                        *                       0 = input  /  1 = output
0000 000E TCHR_OFF      EQU 14  ..15        transfer terminate character
                        *                       -1 = no termination character
0000 0010 TCNT_OFF      EQU 16  ..19        transfer count
0000 0014 TBUF_OFF      EQU 20  ..23        transfer buffer pointer
0000 0018 TBSZ_OFF      EQU 24  ..27        transfer buffer maximum size
0000 001C TEMP_OFF      EQU 28  ..31        transfer empty pointer pointer
0000 0020 TFIL_OFF      EQU 32  ..35        transfer fill pointer
0000 0024 T_PR_OFF      EQU 36  ..39        transfer pointer to eot procedure
                        *                   NIL  no procedure
0000 0028 T_SL_OFF      EQU 40  ..43        transfer eot proc static link
0000 002C T_PM_OFF      EQU 44  ..47        transfer eot proc parameter
0000 0030 T_DMAPRI      EQU 48              dma priority request
                        *
                        *       TRANSFER EQUATES
                        *
0000 0001 TT_INT        EQU 1               interrupt
0000 0002 TT_DMA        EQU 2               DMA
0000 0003 TT_BURST      EQU 3               burst
0000 0004 TT_FHS        EQU 4               fast handshake
```

```
                        ********************************************************************************
                        *
                        *       EXTERNAL REFERANCES for escape
                        *
                        ********************************************************************************
                                  REFA iodeclarations           reference the io lib var. area
                                  REFA sysglobals


                        ********************************************************************************
                        *
                        *       Escape code values
                        *
                        ********************************************************************************
0000 0001 NO_CARD   EQU     1                       no interface
0000 0002 NOT_HPIB  EQU     2                       not an hpib interface
0000 0003 NO_ACTL   EQU     3                       no active controller
0000 0004 NO_DVC    EQU     4                       sc ( not device ) specified
0000 0005 NO_SPACE  EQU     5                       not enough space in the buffer
0000 0006 NO_DATA   EQU     6                       not enough data left in the buffer
0000 0007 TFR_ERR   EQU     7                       tfr error
0000 0008 SC_BUSY   EQU     8                       sc is currently busy
0000 0009 BUF_BUSY  EQU     9                       the buffer is busy
0000 000A TCNTERR   EQU     10                      bad count
0000 000B BADTMO    EQU     11                      bad timeout value
0000 000C NO_DRV    EQU     12                      no driver
0000 000D NO_DMA    EQU     13                      no dma installed
0000 000E NO_WORD   EQU     14                      no word transfers allowed
0000 000F NOT_TALK  EQU     15                      not addressed as talker
0000 0010 NOT_LSTN  EQU     16                      not addressed as listener
0000 0011 TMO_ERR   EQU     17                      timeout
0000 0012 NO_SCTL   EQU     18                      not system controller
0000 0013 BAD_RDS   EQU     19                      bad read status / write control
0000 0014 BAD_SCT   EQU     20                      bad set/clear/test
0000 0015 CRD_DWN   EQU     21                      interface is dead
0000 0016 EOD_SEEN  EQU     22                      end of data has happened
0000 0017 IO_MISC   EQU     23                      misc. error


FFFF FFE6 IOE_ERROR EQU     -26                     io sub system error escape code


FFFF FFBE IOE_RSLT  EQU     IODECLARATIONS-66
FFFF FFBA IOE_SC    EQU     IODECLARATIONS-70


FFFF FFFE ESC_CODE  EQU     SYSGLOBALS-2
FFFF FFF6 RCVR_BLK  EQU     SYSGLOBALS-10
```



```
                        *
                        *       REGISTER USAGE SUMMARY (of utility routines)
                        *
                        *       Global Usage
                        *          a5 -- Pascal Global Base
                        *          a6 -- Pascal Stack Frame
                        *          a7 -- Stack Pointer
                        *          a1 -- Card Address
                        *          a2 -- Driver Attributes ('temp' space)
                        *
                        *       ROUTINE          a0  a3  a4  d0  d1  d2  d3  d4  d5  d6  d7
                        *       ---------------------------------------------------------------
                        *       queue_space      -   -   -   -   -   -   0   -   -   -   -
                        *       queue_empty      -   -   -   -   -   -   -   -   -   -   T
                        *       queue_full       -   -   -   -   -   -   -   -   -   -   T
                        *       inqueue          -   -   G   -   -   I   -   -   -   -   G
                        *       outqueue         -   -   G   -   -   O   -   -   -   -   G
                        *       init_queue       -   -   -   -   -   -   -   -   -   -   -
                        *       check_queue      -   -   -   -   -   -   -   -   -   -   T
                        *       check_dsr_cts    -   -   -   -   -   -   -   -   -   -   T
                        *       wait             -   -   I   -   -   -   -   G   -   L   T
                        *       send             -   -   P   -   -   -   I   L   -   L   T
                        *       get_char         -   -   L   T   -   O   P   L   -   L   L
                        *       wait_send        -   -   L   -   -   -   I   L   -   L   L
                        *       wait_get         -   -   P   L   -   O   L   L   -   L   L
                        *       check_error      -   -   -   -   -   -   -   -   -   -   -
                        *       soft_reset*      -   -   -   -   -   -   -   -   -   T   T
                        *       connect*         -   -   -   -   -   -   -   -   -   L   L
                        *       disconnect       -   -   -   -   -   -   -   -   -   -   -
                        *       rdivu            -   -   -   I   O   -   -   -   -   -   -
                        *       check_xfer_in    -   -   -   -   -   -   -   -   -   -   -
                        *       check_xfer_out   -   -   -   -   -   -   -   -   -   -   -
                        *       clear_xfer+      -   I   -   -   -   -   -   -   -   -   -
                        *       set_xfer         -   I   -   -   -   -   -   -   -   -   -
                        *       dump_buffer      G   I   L   L   -   O   L   L   -   L   L
                        *
                        *       NOTATION (in order of importance)
                        *          O :  output parameter
                        *          I :  input parameter
                        *          G :  used by routine (register has consistent meaning throughout routine
                        *          P :  used to pass parameter to called routines
                        *          T :  used by routine (temporary)
                        *          L :  possible usage by called routines
                        *          - :  not used by routine
                        *
                        *       NOTE:  the registers used by routines to do an ioescape have been
                        *              left out since they do not effect other routines.
                        *
                        *       *This routine calls ABORT_IO which uses other registers not listed
                        *       +This routine calls LOGEOT which uses other registers not listed
```

```
339                     *
340                     *       ROUTINE USAGE SUMMARY
341                     *
342                     *       ROUTINE         CALLS
343                     *       -------         -----
344                     *       queue_space     <none>
345                     *       queue_empty     <none>
346                     *       queue_full      <none>
347                     *       inqueue         <none>
348                     *       outqueue        <none>
349                     *       init_queue      <none>
350                     *       check_queue     <none>
351                     *       check_dsr_cts   <none>
352                     *       wait            check_queue, check_dsr_cts (both indirectly)
353                     *       send            wait, check_dsr_cts
354                     *       get_char        outqueue, queue_space, send
355                     *       wait_send       send, check_error
356                     *       wait_get        wait, check_error, get_char, check_queue
357                     *       check_error     ioescape
358                     *       soft_reset      init_queue, ABORT_IO
359                     *       connect         soft_reset
360                     *       disconnect      <none>
361                     *       rdivu           <none>
362                     *       check_xfer_in   ioescape
363                     *       check_xfer_out  ioescape
364                     *       clear_xfer      <none>
365                     *       set_xfer        <none>
366                     *       dump_buffer     queue_empty, get_char, clear_xfer, LOGEOT
367                     *       ioescape        <none>
368                     *       init            init_queue, ABORT_IO
369                     *       rdb             connect, check_error, check_xfer_in, wait_get
370                     *       wtb             connect, check_error, check_xfer_out, wait_send
371                     *       rdw             connect, check_error, check_xfer_in, wait_get
372                     *       wtw             connect, check_error, check_xfer_out, wait_send
373                     *       rds             queue_empty, get_char, check_error, rdivu, ioescape,
374                     *                       connect
375                     *       wtc             check_error, soft_reset, connect, disconnect, rdivu,
376                     *                       ioescape
377                     *       isr             queue_space, queue_full, inqueue, check_dsr_cts,
378                     *                       send, dump_buffer, LOGEOT
379                     *       tfr             connect, check_error, dump_buffer, set_xfer
380
```



```
382                     *
383                     *
384                     *       ROUTINE         CALLED BY
385                     *       -------         ---------
386                     *       queue_space     get_char, isr
387                     *       queue_empty     dump_buffer, rds (6, 10)
388                     *       queue_full      isr
389                     *       inqueue         isr
390                     *       outqueue        get_char
391                     *       init_queue      init, soft_reset
392                     *       check_queue     wait_get (with wait)
393                     *       check_dsr_cts   isr, send (with and without wait)
394                     *       wait            wait_get, wait_send, send
395                     *       send            isr, wait_send, get_char
396                     *       get_char        rds(6), dump_buffer, wait_get
397                     *       wait_send       wtb, wtw
398                     *       wait_get        rdb, rdw,
399                     *       check_error     rdb, wtb, rdw, wtw, rds(6), wtc(6), tfr,
400                     *                       wait_send, wait_get
401                     *       soft_reset      wtc(14), connect
402                     *       connect         rdb, wtb, rdw, wtw, wtc(12), tfr
403                     *       disconnect      wtc(12)
404                     *       rdivu           rds(3), wtc(3)
405                     *       check_xfer_in   rdb, rdw,
406                     *       check_xfer_out  wtb, wtw
407                     *       clear_xfer      isr, dump_buffer
408                     *       set_xfer        tfr
409                     *       dump_buffer     tfr, isr
410                     *       ioescape        rds, wtc, tfr, check_error, check_xfer_in,
411                     *                       check_xfer_out
412                     *       ABORT_IO        init, soft_reset
413                     *       LOGEOT          isr, dump_buffer
414
415
```

```
                        ********************************************************************************
                        *
                        *       module initialization -- none required.
                        *
                        ********************************************************************************

            0000 0000 RS_RS     EQU *
00000000    4E75                RTS


                        ********************************************************************************
                        *
                        *       98626 card register mnemonics
                        *
                        ********************************************************************************

            0000 0001 RESET_REG         EQU     1               write only
            0000 0001 ID_REG            EQU     1               read only
            0000 0003 INTR_SW           EQU     3               interrupt switches
            0000 0005 BAUD_SW           EQU     5               baud rate switch bank
            0000 0007 LINE_SW           EQU     7               line characteristic switches

                        *
                        *       UART registers
                        *

            0000 0011 DATA              EQU     17              receive/transmit buffer (dlab=0)
            0000 0013 INTR_EN           EQU     19              interrupt enable register(dlab=0)
            0000 0011 DIV0              EQU     17              divisor latch (LSB)    (DLAB=1)
            0000 0013 DIV1              EQU     19              divisor latch (MSB)    (DLAB=1)
            0000 0015 INTR_ID           EQU     21              interrupt identification
            0000 0017 LINE_CONT         EQU     23              line control register
            0000 0019 MODEM_CONT        EQU     25              modem control register
            0000 001B LINE_STAT         EQU     27              line status register
            0000 001D MODEM_STAT        EQU     29              modem status register
```



```
                        ********************************************************************************
                        *
                        *       ATTRIBUTE space offset mnemonics
                        *         (do not mix -- word boundary problems)
                        *         the word address is assumed to be EVEN
                        *         starting at AVAIL_OFF
                        *
                        ********************************************************************************
                        *                                       size
                        *                                       ----
            0000 0034 S_ERROR           EQU     AVAIL_OFF       4  pending error number
            0000 0038 IN_ISR            EQU     S_ERROR+4       1
            0000 0039 XIN_ACT           EQU     IN_ISR+1        1
            0000 003A CONNECTED         EQU     XIN_ACT+1       1
            0000 003B MODEM_ON          EQU     CONNECTED+1     1
            0000 003C RECEIVING         EQU     MODEM_ON+1      1
            0000 003D XMITTING          EQU     RECEIVING+1     1
            0000 003E S_MODEM           EQU     XMITTING+1      1  modem status copy
            0000 003F S_LINE            EQU     S_MODEM+1       1  line status copy
            0000 0040 S_HANDSH          EQU     S_LINE+1        1  contains current handshake
            0000 0041 XON_CHAR          EQU     S_HANDSH+1      1
            0000 0042 XOFF_CHAR         EQU     XON_CHAR+1      1
            0000 0043 ENQ_CHAR          EQU     XOFF_CHAR+1     1
            0000 0044 ACK_CHAR          EQU     ENQ_CHAR+1      1
            0000 0045 CONV_CHAR         EQU     ACK_CHAR+1      1
            0000 0046 Q_DESCRIPT        EQU     CONV_CHAR+1
            0000 0046 Q_SIZE            EQU     Q_DESCRIPT      2
            0000 0048 Q_IN              EQU     Q_DESCRIPT+2    2
            0000 004A Q_OUT             EQU     Q_DESCRIPT+4    2
            0000 004C Q_BUFFER          EQU     Q_DESCRIPT+6    136 the rest is internal buffer
                        *                                       ---
                        *                                       160  total space used


                        ********************************************************************************
                        *
                        *       constants (mnemonics)
                        *
                        ********************************************************************************

            0000 00A0 TEMP_SIZE         EQU     160             Pascal declaration of size of misc. space
            0000 0088 BUFFER_SIZE       EQU     TEMP_SIZE-Q_BUFFER+AVAIL_OFF
            0000 0011 DC1               EQU     17              ASCII CHARACTER 17
            0000 0013 DC3               EQU     19              ASCII CHARACTER 19
            0000 0005 ENQ               EQU     5               ASCII CHARACTER 5
            0000 0006 ACK               EQU     6               ASCII CHARACTER 6
            0000 005F UNDERSCORE        EQU     95              ASCII CHARACTER 95
            0000 013A OVERRUN_ERROR     EQU     314             receive buffer overflow error
            0000 0050 ACK_SIZE          EQU     80              hysteresis for ENQ/ACK handshake
            0000 0028 XOFF_SIZE         EQU     40              when to send XOFF
            0000 0060 XON_SIZE          EQU     BUFFER_SIZE-XOFF_SIZE
            0000 0013 REG_MAX           EQU     19              maximum control/status register number
```

```
                        ********************************************************************************
                        *
                        *       driver initialization
                        *
                        ********************************************************************************

            0000 0002 RS_RS_INIT      EQU *

                        *
                        *   Pascal interface overhead
                        *
    00000002 205F               MOVEA.L (SP)+,A0        * get return address
    00000004 245F               MOVEA.L (SP)+,A2        * get temp address
    00000006 226A 0020          MOVEA.L C_ADR(A2),A1    * get card address
    0000000A 4850               PEA     (A0)            * restore return address
                        *
                        *   Assembler initialize entry point
                        *       ON ENTRY:  A1  and  A2  are set to card address and
                        *                  temp space address (respectively)
                        *

            0000 000C ASM_INIT        EQU *

                        *
                        *   Stop transfers and Reset the card
                        *
    0000000C 4EB9 0000          JSR     ABORT_IO        * stop transfers
             0000
                        *                                    -- RESET CARD --
    00000012 50E9 0001          ST      RESET_REG(A1)   *  write to reg 1
    00000016 7020               MOVEQ   #32,D0          *  wait 25 us
    00000018 51C8 FFFE          DBRA    D0,*            *  ( not yet timed -- #32 was initial est.)

                        *
                        *       global attribute initialization
                        *
                        *                                    -- init queue descriptors --
    0000001C 6100 09B2          BSR     INIT_QUEUE
                        *                                    -- init pseudo registers --
    00000020 422A 0039          CLR.B   XIN_ACT(A2)
    00000024 422A 0038          CLR.B   IN_ISR(A2)
    00000028 42AA 0034          CLR.L   S_ERROR(A2)
    0000002C 422A 003E          CLR.B   S_MODEM(A2)
    00000030 422A 003F          CLR.B   S_LINE(A2)
    00000034 422A 003A          CLR.B   CONNECTED(A2)
    00000038 422A 003B          CLR.B   MODEM_ON(A2)
                        *                                    -- set flags --
    0000003C 157C 0001          MOVE.B  #1,RECEIVING(A2)
             003C
    00000042 157C 0001          MOVE.B  #1,XMITTING(A2)
             003D
                        *                                    -- default characters --
    00000048 157C 0011          MOVE.B  #DC1,XON_CHAR(A2)
             0041
    0000004E 157C 0013          MOVE.B  #DC3,XOFF_CHAR(A2)
             0042
```



```
    00000054 157C 0005          MOVE.B  #ENQ,ENQ_CHAR(A2)
             0043
    0000005A 157C 0006          MOVE.B  #ACK,ACK_CHAR(A2)
             0044
    00000060 157C 005F          MOVE.B  #UNDERSCORE,CONV_CHAR(A2)
             0045

                        *
                        *   Set defaults from the switches (done after attribute initialization
                        *                just in case the card is not completely reset already)
                        *
                        *                                    -- set baud rate from switches --
    00000066 1029 0005          MOVE.B  BAUD_SW(A1),D0  *   read switches
    0000006A E308               LSL.B   #1,D0           *   multiply by two to get to words
    0000006C 0200 001E          ANDI.B  #$1E,D0         *   mask
    00000070 4880               EXT     D0              *
                        *
    00000072 41FA 002C          LEA     BAUD,A0         *   a0 :=  pointer to baud table
    00000076 D0C0               ADDA    D0,A0           *              entry
                        *
    00000078 08E9 0007          BSET    #7,LINE_CONT(A1) *   set DLAB to get to divisor latches
             0017
    0000007E 1358 0013          MOVE.B  (A0)+,DIV1(A1)  *
    00000082 1350 0011          MOVE.B  (A0),DIV0(A1)   *

                        *                                    -- SET LINE CHARACTERISTICS --
    00000086 1029 0007          MOVE.B  LINE_SW(A1),D0  *  get handshake and line status defaults
    0000008A 1200               MOVE.B  D0,D1           *
    0000008C 0200 003F          ANDI.B  #$3F,D0         *  mask and set default line status
    00000090 1340 0017          MOVE.B  D0,LINE_CONT(A1) *    (also restores dlab)
                        *                                    -- SAVE HANDSHAKE --
    00000094 0201 00C0          ANDI.B  #$C0,D1         *  handshake is top two bits
    00000098 EA09               LSR.B   #5,D1           *  adjust it (for jump tables)
    0000009A 1541 0040          MOVE.B  D1,S_HANDSH(A2) *  save it

    0000009E 4E75               RTS
                        *
                        *       baud rate table (for switches)
                        *
    000000A0 0C00       BAUD    DC.W    3072            *    50 Baud
    000000A2 0800               DC.W    2048            *    75
    000000A4 0574               DC.W    1396            *   110
    000000A6 0476               DC.W    1142            *   134.5
    000000A8 0400               DC.W    1024            *   150
    000000AA 0300               DC.W     768            *   200
    000000AC 0200               DC.W     512            *   300
    000000AE 0100               DC.W     256            *   600
    000000B0 0080               DC.W     128            *  1200
    000000B2 0055               DC.W      85            *  1800
    000000B4 0040               DC.W      64            *  2400
    000000B6 002B               DC.W      43            *  3600
    000000B8 0020               DC.W      32            *  4800
    000000BA 0015               DC.W      21            *  7200
    000000BC 0010               DC.W      16            *  9600
    000000BE 0008               DC.W       8            * 19200
```

```
                        ********************************************************************************
                        *
                        *       read byte
                        *
                        ********************************************************************************

              0000 00C0 RS_RS_RDB       EQU *


                        *
                        *   Pascal interface overhead
                        *
000000C0 205F                   MOVEA.L (SP)+,A0                get return address
000000C2 265F                   MOVEA.L (SP)+,A3                get VAR address
000000C4 245F                   MOVEA.L (SP)+,A2                get temp address
000000C6 226A 0020              MOVEA.L C_ADR(A2),A1            get card address
000000CA 4850                   PEA     (A0)                    restore return address
                        *
                        *   Card overhead (any of the three can do an ioescape)
                        *
000000CC 6100 0728              BSR     CONNECT                 make sure the card is active
000000D0 6100 07A2              BSR     CHECK_ERROR             check for errors saved by ISRs
000000D4 6100 06DE              BSR     CHECK_XFER_IN           make sure no input transfers active

000000D8 6100 07C8              BSR     WAIT_GET                get character with wait
000000DC 1682                   MOVE.B  D2,(A3)                 return the character
000000DE 4E75                   RTS
```



```
                        ********************************************************************************
                        *
                        *       write byte
                        *
                        ********************************************************************************

              0000 00E0 RS_RS_WTB       EQU *


                        *
                        *   Pascal interface overhead
                        *
000000E0 205F                   MOVEA.L (SP)+,A0                get return address
000000E2 161F                   MOVE.B  (SP)+,D3                get char to be written
000000E4 245F                   MOVEA.L (SP)+,A2                get temp address
000000E6 226A 0020              MOVEA.L C_ADR(A2),A1            get card address
000000EA 4850                   PEA     (A0)                    restore return address
                        *
                        *   Card overhead
                        *
000000EC 6100 0708              BSR     CONNECT                 autoconnect
000000F0 6100 0782              BSR     CHECK_ERROR             check for errors found by ISRs
000000F4 6100 06C6              BSR     CHECK_XFER_OUT          make sure no output transfers are active

000000F8 6100 079A              BSR     WAIT_SEND               send the character
000000FC 4E75                   RTS
```

```
                        ********************************************************************************
                        *
                        *       read word
                        *
                        ********************************************************************************

              0000 00FE RS_RS_RDW       EQU *

    000000FE 205F               MOVEA.L (SP)+,A0                get return address
    00000100 265F               MOVEA.L (SP)+,A3                get VAR address
    00000102 245F               MOVEA.L (SP)+,A2                get temp address
    00000104 226A 0020          MOVEA.L C_ADR(A2),A1            get card address
    00000108 4850               PEA     (A0)                    restore return address
                        *
                        *   Card overhead (any of the three can do an ioescape)
                        *
    0000010A 6100 06EA          BSR     CONNECT                 make sure the card is active
    0000010E 6100 0764          BSR     CHECK_ERROR             check for errors saved by ISRs
    00000112 6100 06A0          BSR     CHECK_XFER_IN           make sure no input transfers are active

    00000116 6100 078A          BSR     WAIT_GET                get character with wait
    0000011A E14A               LSL.W   #8,D2                   shift first character
    0000011C 6100 0784          BSR     WAIT_GET                get second character
    00000120 3682               MOVE.W  D2,(A3)                 return the word
    00000122 4E75               RTS
```



```
                        ********************************************************************************
                        *
                        *       write word
                        *
                        ********************************************************************************

              0000 0124 RS_RS_WTW       EQU *

    00000124 205F               MOVEA.L (SP)+,A0                get return address
    00000126 361F               MOVE.W  (SP)+,D3                get word to be written
    00000128 245F               MOVEA.L (SP)+,A2                get temp address
    0000012A 228A 0020          MOVEA.L C_ADR(A2),A1            get card address
    0000012E 4850               PEA     (A0)                    restore return address
                        *
                        *   Card overhead (any of the three can do an ioescape)
                        *
    00000130 6100 06C4          BSR     CONNECT                 make sure the card is active
    00000134 6100 073E          BSR     CHECK_ERROR             check for errors saved by ISRs
    00000138 6100 0682          BSR     CHECK_XFER_OUT          make sure no output transfers are active

    0000013C E05B               ROR.W   #8,D3                   position the first character
    0000013E 6100 0754          BSR     WAIT_SEND               send it
    00000142 E05B               ROR.W   #8,D3                   position the second character
    00000144 6100 074E          BSR     WAIT_SEND               send it

    00000148 4E75               RTS
```

```
                        ********************************************************************************
                        *
                        *       read status
                        *-------------------------------------------------------------------------------
                        *       CONVENTION:  A3 -- place to put the result (word sized)
                        *
                        ********************************************************************************

              0000 014A RS_RS_RDS       EQU *
                        *
                        *   Pascal Interface Overhead
                        *

 0000014A 205F                  MOVEA.L (SP)+,A0                get return address
 0000014C 265F                  MOVEA.L (SP)+,A3                get VAR address
 0000014E 321F                  MOVE.W  (SP)+,D1                get register number
 00000150 245F                  MOVEA.L (SP)+,A2                get temp address
 00000152 226A 0020             MOVEA.L C_ADR(A2),A1            get card address
 00000156 4850                  PEA     (A0)                    restore return address

                        *
                        *   Check for legal registers and jump to correct register handler
                        *

 00000158 4A41                  TST.W   D1                      check for negative register number
 0000015A 6D3C                  BLT.S   STS_ERROR               if so goto common ioescape routine
 0000015C B27C 0013             CMP.W   #REG_MAX,D1             check for too large register number
 00000160 6E36                  BGT.S   STS_ERROR

 00000162 4253                  CLR.W   (A3)                    clear the top half of the return word
 00000164 528B                  ADDQ.L  #1,A3                   point a3 to lower half byte of return word

 00000166 E349                  LSL.W   #1,D1                   get ready for (word) table jump
 00000168 323B 1006             MOVE.W  STS_TABLE(D1),D1        get offset from status table
 0000016C 4EFB 1002             JMP     STS_TABLE(D1)           do the indexed jump

              0000 0170 STS_TABLE       EQU *
 00000170 002E                  DC.W    STS_0-STS_TABLE         ID register
 00000172 0034                  DC.W    STS_1-STS_TABLE         Interrupt status register
 00000174 003A                  DC.W    STS_2-STS_TABLE         Busy bits register
 00000176 006C                  DC.W    STS_3-STS_TABLE         Baud rate
 00000178 00D4                  DC.W    STS_4-STS_TABLE         Character control register
 0000017A 00E8                  DC.W    STS_5-STS_TABLE         Modem control register
 0000017C 00EE                  DC.W    STS_6-STS_TABLE         Data in register
 0000017E 0112                  DC.W    STS_7-STS_TABLE         Optional circuits register
 00000180 011C                  DC.W    STS_8-STS_TABLE         Interrupt Enable Mask register
 00000182 0122                  DC.W    STS_9-STS_TABLE         Interrupt Cause register
 00000184 0128                  DC.W    STS_10-STS_TABLE        UART Status register
 00000186 0152                  DC.W    STS_11-STS_TABLE        Modem Status register
 00000188 0172                  DC.W    STS_12-STS_TABLE        Connect/Disconnect register
 0000018A 0178                  DC.W    STS_13-STS_TABLE        Hardware handshake register
 0000018C 017E                  DC.W    STS_14-STS_TABLE        Error status register
 0000018E 0196                  DC.W    STS_15-STS_TABLE        Current Xon Character
 00000190 019C                  DC.W    STS_16-STS_TABLE        Current Xoff Character
 00000192 01A2                  DC.W    STS_17-STS_TABLE        Current ENQ Character
 00000194 01A8                  DC.W    STS_18-STS_TABLE        Current ACK Character
 00000196 01AE                  DC.W    STS_19-STS_TABLE        Current FE/PE convert Character
```



```

              0000 0198 STS_ERROR       EQU *
 00000198 7013                  MOVEQ   #BAD_RDS,D0
 0000019A 6000 062E             BRA     IOESCAPE                error number is passed in d0
                        *-------------------------------------------------------------------------------
              0000 019E STS_0           EQU *                   -- ID register --
 0000019E 16A9 0001             MOVE.B  ID_REG(A1),(A3)         get id from card
 000001A2 4E75                  RTS
                        *-------------------------------------------------------------------------------
              0000 01A4 STS_1           EQU *                   -- Interrupt Status --
 000001A4 16A9 0003             MOVE.B  INTR_SW(A1),(A3)        get result from card
 000001A8 4E75                  RTS
                        *-------------------------------------------------------------------------------
              0000 01AA STS_2           EQU *                   -- "Busy Bits" --
 000001AA 1E29 0003             MOVE.B  INTR_SW(A1),D7          --> interrupt enabled bit (1)
 000001AE CE3C 0080             AND.B   #$80,D7
 000001B2 EC0F                  LSR.B   #6,D7                   move it to correct position

 000001B4 4AAA 0024             TST.L   BUFI_OFF(A2)            --> transfer active bit (0)
 000001B8 6606                  BNE.S   SET_BIT_0
 000001BA 4AAA 0028             TST.L   BUFO_OFF(A2)
 000001BE 6704                  BEQ.S   DO_BIT_4                neither transfer is active

              0000 01C0 SET_BIT_0       EQU *                   set transfer active bit
 000001C0 08C7 0000             BSET    #0,D7

              0000 01C4 DO_BIT_4        EQU *                   --> not transmitting bit (4)
 000001C4 4A2A 003D             TST.B   XMITTING(A2)
 000001C8 6604                  BNE.S   DO_BIT_5
 000001CA 08C7 0004             BSET    #4,D7

              0000 01CE DO_BIT_5        EQU *                   --> not receiving bit (5)
 000001CE 4A2A 003C             TST.B   RECEIVING(A2)
 000001D2 6604                  BNE.S   END_STS_2
 000001D4 08C7 0005             BSET    #5,D7

              0000 01D8 END_STS_2       EQU *                   store away result
 000001D8 1687                  MOVE.B  D7,(A3)
 000001DA 4E75                  RTS
                        *-------------------------------------------------------------------------------
              0000 01DC STS_3           EQU *                   -- Baud rate --
 000001DC 538B                  SUBQ.L  #1,A3                   this routine returns a word
                        *
                        *   Get divisor (a critical section since it uses DLAB)
                        *
 000001DE 1E29 0003             MOVE.B  INTR_SW(A1),D7          save card interrupt status
 000001E2 4229 0003             CLR.B   INTR_SW(A1)             disable interrupts

 000001E6 08E9 0007             BSET    #7,LINE_CONT(A1)        get access to divisor latches
          0017
 000001EC 1029 0013             MOVE.B  DIV1(A1),D0             get upper half of divisor
 000001F0 E148                  LSL.W   #8,D0
 000001F2 1029 0011             MOVE.B  DIV0(A1),D0             get lower half of divisor
 000001F6 08A9 0007             BCLR    #7,LINE_CONT(A1)        reset DLAB so normal operation can resume
          0017

 000001FC 1347 0003             MOVE.B  D7,INTR_SW(A1)          restore interrupts
```

```
                        *
                        *   Check for special divisors which division is inexact
                        *
00000200 4A40                   TST.W   D0                      - infinite baud rate ? -
00000202 6602                     BNE.S   IS85
00000204 4E75                     RTS                           return zero baud rate
00000206 B07C 0055 IS85         CMP.W   #85,D0                  - 1800 baud ? -
0000020A 6606                     BNE.S   IS77
0000020C 36BC 0708                MOVE.W  #1800,(A3)
00000210 4E75                     RTS
00000212 B07C 004D IS77         CMP.W   #77,D0                  - 2000 baud ? -
00000216 6606                     BNE.S   IS43
00000218 36BC 07D0                MOVE.W  #2000,(A3)
0000021C 4E75                     RTS
0000021E B07C 002B IS43         CMP.W   #43,D0                  - 3600 baud ? -
00000222 6606                     BNE.S   IS21
00000224 36BC 0E10                MOVE.W  #3600,(A3)
00000228 4E75                     RTS
0000022A B07C 0015 IS21         CMP.W   #21,D0                  - 7200 baud ? -
0000022E 6606                     BNE.S   REGULAR
00000230 36BC 1C20                MOVE.W  #7200,(A3)
00000234 4E75                     RTS
            0000 0236 REGULAR           EQU *
                        *
                        *   Compute baud rate by:  baud_rate = (freq/16) / divisor
                        *
00000236 223C 0002              MOVE.L  #153600,D1
         5800
0000023C 6100 05A2              BSR     RDIVU                   do the division
00000240 3681                   MOVE.W  D1,(A3)                 store away the answer (d1)
00000242 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 0244 STS_4             EQU *                   -- Character control --
00000244 1E29 0017              MOVE.B  LINE_CONT(A1),D7        get line control
00000248 CE3C 003F              AND.B   #$3F,D7                 remove top two bits

0000024C 1C2A 0040              MOVE.B  S_HANDSH(A2),D6         get handshake
00000250 EB0E                   LSL.B   #5,D6                   move it to top two bits

00000252 8E06                   OR.B    D6,D7                   combine to form register result
00000254 1687                   MOVE.B  D7,(A3)
00000256 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 0258 STS_5             EQU *                   -- Modem control --
00000258 16A9 0019              MOVE.B  MODEM_CONT(A1),(A3)     read directly from the UART
0000025C 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 025E STS_6             EQU *                   -- Data in --
0000025E 6100 0614              BSR     CHECK_ERROR             check for errors trapped by ISRs

00000262 6100 077C              BSR     QUEUE_EMPTY             read from buffer if not empty
00000266 6714                   BEQ.S   READ_UART               else read directly from UART

00000268 1A29 0003              MOVE.B  INTR_SW(A1),D5          save interrupt state
0000026C 4229 0003              CLR.B   INTR_SW(A1)             disable card interrupts for critical section

00000270 6100 0642              BSR     GET_CHAR                (get character with handshake)
```



```
00000274 1682                   MOVE.B  D2,(A3)

00000276 1345 0003              MOVE.B  D5,INTR_SW(A1)          restore interrupt state
0000027A 4E75                   RTS
            0000 027C READ_UART         EQU *
0000027C 16A9 0011              MOVE.B  DATA(A1),(A3)
00000280 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 0282 STS_7             EQU *                   -- Optional circuits --
00000282 1E29 0005              MOVE.B  BAUD_SW(A1),D7          read from the card hardware
00000286 E80F                   LSR.B   #4,D7                   right justify (and get rid of baud info)
00000288 1687                   MOVE.B  D7,(A3)
0000028A 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 028C STS_8             EQU *                   -- interrupt enable mask --
0000028C 16A9 0013              MOVE.B  INTR_EN(A1),(A3)        read directly from the UART
00000290 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 0292 STS_9             EQU *                   -- interrupt cause --
00000292 16A9 0015              MOVE.B  INTR_ID(A1),(A3)        read directly from the UART
00000296 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 0298 STS_10            EQU *                   -- UART status --
00000298 1E29 0003              MOVE.B  INTR_SW(A1),D7          save card interrupt condition
0000029C 4229 0003              CLR.B   INTR_SW(A1)             disable interrupts for critical section

000002A0 1C2A 003F              MOVE.B  S_LINE(A2),D6           get accumulated line status
000002A4 422A 003F              CLR.B   S_LINE(A2)              reset it since it is read destructive

000002A8 1347 0003              MOVE.B  D7,INTR_SW(A1)          restore interrupts (end critical section)

000002AC CC3C 001E              AND.B   #$1E,D6                 only use the read destructive bits
000002B0 8C29 001B              OR.B    LINE_STAT(A1),D6        combine it with the current status

000002B4 6100 072A              BSR     QUEUE_EMPTY             use internal buffer to determine bit 0
000002B8 6704                   BEQ.S   DONT_SET
000002BA 08C6 0000              BSET    #0,D6                   set receive buffer full bit
            0000 02BE DONT_SET          EQU *
000002BE 1686                   MOVE.B  D6,(A3)
000002C0 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 02C2 STS_11            EQU *                   -- modem status --
000002C2 1E29 0003              MOVE.B  INTR_SW(A1),D7          save card interrupt condition
000002C6 4229 0003              CLR.B   INTR_SW(A1)             disable interrupts for critical section

000002CA 1C2A 003E              MOVE.B  S_MODEM(A2),D6          get accumulated copy of modem status
000002CE 422A 003E              CLR.B   S_MODEM(A2)             clear it since it is read destructive

000002D2 1347 0003              MOVE.B  D7,INTR_SW(A1)          restore interrupts (end critical section)

000002D6 CC3C 000F              AND.B   #$0F,D6                 only use the read destructive bits
000002DA 8C29 001D              OR.B    MODEM_STAT(A1),D6       combine it with the current status
000002DE 1686                   MOVE.B  D6,(A3)
000002E0 4E75                   RTS
                        *-------------------------------------------------------------------------------
            0000 02E2 STS_12            EQU *                   -- connect/disconnect --
000002E2 16AA 003A              MOVE.B  CONNECTED(A2),(A3)      get the pseudo-register
```

```
 951    000002E6 4E75              RTS
 952                       *-------------------------------------------------------------------------------
 953              0000 02E8 STS_13          EQU *                       -- hardware handshake register --
 954    000002E8 16AA 003B         MOVE.B  MODEM_ON(A2),(A3)
 955    000002EC 4E75              RTS
 956                       *-------------------------------------------------------------------------------
 957              0000 02EE STS_14          EQU *                       -- current error status --
 958    000002EE 1E29 0003         MOVE.B  INTR_SW(A1),D7              save interrupt state
 959    000002F2 4229 0003         CLR.B   INTR_SW(A1)                 disable interrupts
 960
 961    000002F6 376A 0036         MOVE.W  S_ERROR+2(A2),-1(A3)        get (lower word of) the error
                  FFFF
 962                       *                                           this is a word register!
 963    000002FC 42AA 0034         CLR.L   S_ERROR(A2)                 the read is destructive
 964
 965    00000300 1347 0003         MOVE.B  D7,INTR_SW(A1)              restore interrupt state
 966    00000304 4E75              RTS
 967                       *-------------------------------------------------------------------------------
 968              0000 0306 STS_15          EQU *                       -- Current Xon character --
 969    00000306 16AA 0041         MOVE.B  XON_CHAR(A2),(A3)
 970    0000030A 4E75              RTS
 971                       *-------------------------------------------------------------------------------
 972              0000 030C STS_16          EQU *                       -- Current Xoff character --
 973    0000030C 16AA 0042         MOVE.B  XOFF_CHAR(A2),(A3)
 974    00000310 4E75              RTS
 975                       *-------------------------------------------------------------------------------
 976              0000 0312 STS_17          EQU *                       -- Current ENQ character --
 977    00000312 16AA 0043         MOVE.B  ENQ_CHAR(A2),(A3)
 978    00000316 4E75              RTS
 979                       *-------------------------------------------------------------------------------
 980              0000 0318 STS_18          EQU *                       -- Current ACK character --
 981    00000318 16AA 0044         MOVE.B  ACK_CHAR(A2),(A3)
 982    0000031C 4E75              RTS
 983                       *-------------------------------------------------------------------------------
 984              0000 031E STS_19          EQU *                       -- Current FE/PE convert character --
 985    0000031E 16AA 0045         MOVE.B  CONV_CHAR(A2),(A3)
 986    00000322 4E75              RTS
```



```
 989                       ********************************************************************************
 990                       *
 991                       *       write control
 992                       *-------------------------------------------------------------------------------
 993                       *       CONVENTION:  D0.W -- value of the control
 994                       *
 995                       ********************************************************************************
 996
 997              0000 0324 RS_RS_WTC       EQU *
 998
 999                       *
1000                       *   Pascal Interface Overhead
1001                       *
1002
1003    00000324 205F              MOVEA.L (SP)+,A0                    get return address
1004    00000326 301F              MOVE.W  (SP)+,D0                    get value
1005    00000328 321F              MOVE.W  (SP)+,D1                    get register number
1006    0000032A 245F              MOVEA.L (SP)+,A2                    get temp address
1007    0000032C 226A 0020         MOVEA.L C_ADR(A2),A1                get card address
1008    00000330 4850              PEA     (A0)                        restore return address
1009
1010                       *
1011                       *   Check for legal registers and jump to correct register handler
1012                       *
1013
1014    00000332 4A41              TST.W   D1                          check for negative register number
1015    00000334 6B38              BMI.S   CONT_ERROR                  if so goto common ioescape routine
1016    00000336 B27C 0013         CMP.W   #REG_MAX,D1                 check for too large register number
1017    0000033A 6E32              BGT.S   CONT_ERROR
1018
1019    0000033C E349              LSL.W   #1,D1                       get ready for (word) table jump
1020    0000033E 323B 1006         MOVE.W  CONT_TABLE(D1),D1           get offset from status table
1021    00000342 4EFB 1002         JMP     CONT_TABLE(D1)              do the indexed jump
1022
1023              0000 0346 CONT_TABLE       EQU *
1024    00000346 002E              DC.W    CONT_0-CONT_TABLE           Reset
1025    00000348 0036              DC.W    CONT_1-CONT_TABLE           Break
1026    0000034A 0028              DC.W    CONT_ERROR-CONT_TABLE       Register 2 Undefined
1027    0000034C 005C              DC.W    CONT_3-CONT_TABLE           Baud rate
1028    0000034E 0098              DC.W    CONT_4-CONT_TABLE           Character control register
1029    00000350 00CC              DC.W    CONT_5-CONT_TABLE           Modem control register
1030    00000352 00D2              DC.W    CONT_6-CONT_TABLE           Data out register
1031    00000354 00DC              DC.W    CONT_7-CONT_TABLE           Optional circuits register
1032    00000356 0028              DC.W    CONT_ERROR-CONT_TABLE       Register 8 Undefined
1033    00000358 0028              DC.W    CONT_ERROR-CONT_TABLE       Register 9 Undefined
1034    0000035A 0028              DC.W    CONT_ERROR-CONT_TABLE       Register 10 Undefined
1035    0000035C 0028              DC.W    CONT_ERROR-CONT_TABLE       Register 11 Undefined
1036    0000035E 00E4              DC.W    CONT_12-CONT_TABLE          Connect/Disconnect register
1037    00000360 00F8              DC.W    CONT_13-CONT_TABLE          Hardware handshake register
1038    00000362 0108              DC.W    CONT_14-CONT_TABLE          Soft reset register
1039    00000364 0112              DC.W    CONT_15-CONT_TABLE          Redefine Xon Character
1040    00000366 0118              DC.W    CONT_16-CONT_TABLE          Redefine Xoff Character
1041    00000368 011E              DC.W    CONT_17-CONT_TABLE          Redefine ENQ Character
1042    0000036A 0124              DC.W    CONT_18-CONT_TABLE          Redefine ACK Character
1043    0000036C 012A              DC.W    CONT_19-CONT_TABLE          Redefine FE/PE convert Character
1044
1045              0000 036E CONT_ERROR      EQU *
```

```
1046    0000036E 7013              MOVEQ   #BAD_RDS,D0
1047    00000370 6000 0458         BRA     IOESCAPE                error number is passed in d0
1048                       *-------------------------------------------------------------------------------
1049             0000 0374 CONT_0          EQU *                     -- reset --
1050    00000374 4A40              TST.W   D0
1051    00000376 6600 FC94         BNE     ASM_INIT                initialize if any bit is set
1052    0000037A 4E75              RTS
1053                       *-------------------------------------------------------------------------------
1054             0000 037C CONT_1          EQU *                     -- send break --
1055    0000037C 4A40              TST.W   D0
1056    0000037E 6720              BEQ.S   EXIT_C1                 no-op if control value is zero
1057                       *
1058                       *   set and hold break for 400ms
1059                       *
1060    00000380 08E9 0006         BSET    #6,LINE_CONT(A1)        set the break bit in UART
                0017
1061    00000386 203C 0002         MOVE.L  #145600,D0              set 400 ms loop counter
                38C0
1062             0000 038C WAIT_BREAK      EQU *
1063    0000038C 5380              SUBQ.L  #1,D0
1064    0000038E 6AFC              BPL.S   WAIT_BREAK              hold the break until 400ms passed
1065                       *
1066                       *   release break and wait 60ms for break to clear
1067                       *
1068    00000390 08A9 0006         BCLR    #6,LINE_CONT(A1)        clear the break bit in UART
                0017
1069    00000396 203C 0000         MOVE.L  #21840,D0
                5550
1070             0000 039C WAIT_BREAK2     EQU *
1071    0000039C 5380              SUBQ.L  #1,D0
1072    0000039E 6AFC              BPL.S   WAIT_BREAK2             wait until counter is negative
1073    000003A0 4E75              EXIT_C1 RTS
1074                       *-------------------------------------------------------------------------------
1075             0000 03A2 CONT_3          EQU *                     -- Baud rate --
1076                       *
1077                       *   check for overflow -- a baud rate with a resulting divisor more than
1078                       *                         sixteen bits long.
1079                       *   (underflow is not checked because it cannot be generated with a word
1080                       *    length baud rate)
1081                       *
1082    000003A2 B07C 0005         CMP.W   #5,D0                   5 is the lowest baud rate possible
1083    000003A6 6C00 0008         BGE     CALC_DIV
1084
1085    000003AA 7017              MOVEQ   #IO_MISC,D0             value out of range -- io misc error
1086    000003AC 6000 041C         BRA     IOESCAPE
1087                       *
1088                       *   calculate divisor :  div = (freq/16) / baud_rate
1089                       *
1090             0000 03B0 CALC_DIV        EQU *
1091    000003B0 223C 0002         MOVE.L  #153600,D1              d1 := freq/16
                5800
1092    000003B6 6100 0428         BSR     RDIVU                   d1.w := d1.l / d0.w ( freq/16 / baud )
1093                       *
1094                       *   move the divisor to hardware (critical section: uses DLAB)
1095                       *
1096    000003BA 1E29 0003         MOVE.B  INTR_SW(A1),D7          save card interrupt state
1097    000003BE 4229 0003         CLR.B   INTR_SW(A1)             disable card interrupts
```



```
1098
1099    000003C2 08E9 0007         BSET    #7,LINE_CONT(A1)        set DLAB to get access to divisor latches
                0017
1100    000003C8 1341 0011         MOVE.B  D1,DIV0(A1)             set lower half of divisor latch
1101    000003CC E049              LSR.W   #8,D1
1102    000003CE 1341 0013         MOVE.B  D1,DIV1(A1)             set upper half of divisor latch
1103    000003D2 08A9 0007         BCLR    #7,LINE_CONT(A1)        clear DLAB for normal use
                0017
1104
1105    000003D8 1347 0003         MOVE.B  D7,INTR_SW(A1)          restore card interrupt state
1106
1107    000003DC 4E75              RTS
1108                       *-------------------------------------------------------------------------------
1109             0000 03DE CONT_4          EQU *                     -- Character Control --
1110    000003DE 1E00              MOVE.B  D0,D7                   save a copy of control value
1111
1112    000003E0 C03C 003F         AND.B   #$3F,D0                 use only bottom 6 bits
1113    000003E4 1340 0017         MOVE.B  D0,LINE_CONT(A1)        for line control
1114
1115    000003E8 CE3C 00C0         AND.B   #$C0,D7                 handshake is top two bits
1116    000003EC EA0F              LSR.B   #5,D7                   shift it for later use
1117    000003EE BE2A 0040         CMP.B   S_HANDSH(A2),D7         if handshake changed
1118    000003F2 671C              BEQ.S   EXIT_C4
1119
1120    000003F4 1C29 0003         MOVE.B  INTR_SW(A1),D6          save interrupt state
1121    000003F8 4229 0003         CLR.B   INTR_SW(A1)             disable interrupts
1122
1123    000003FC 1547 0040         MOVE.B  D7,S_HANDSH(A2)         save new handshake
1124    00000400 157C 0001         MOVE.B  #1,RECEIVING(A2)        \
                003C
1125    00000406 157C 0001         MOVE.B  #1,XMITTING(A2)         /   reset the handshake flags
                003D
1126
1127    0000040C 1346 0003         MOVE.B  D6,INTR_SW(A1)          restore interrupt state
1128    00000410 4E75              EXIT_C4 RTS
1129                       *-------------------------------------------------------------------------------
1130             0000 0412 CONT_5          EQU *                     -- modem control --
1131    00000412 1340 0019         MOVE.B  D0,MODEM_CONT(A1)       write directly to UART
1132    00000416 4E75              RTS
1133                       *-------------------------------------------------------------------------------
1134             0000 0418 CONT_6          EQU *                     -- Data out --
1135    00000418 6100 045A         BSR     CHECK_ERROR             check for errors trapped by ISRs
1136
1137    0000041C 1340 0011         MOVE.B  D0,DATA(A1)             write directly to UART
1138    00000420 4E75              RTS
1139                       *-------------------------------------------------------------------------------
1140             0000 0422 CONT_7          EQU *                     -- Optional Circuits --
1141    00000422 E908              LSL.B   #4,D0                   left justify
1142    00000424 1340 0005         MOVE.B  D0,BAUD_SW(A1)          write directly to card hardware
1143    00000428 4E75              RTS
1144                       *-------------------------------------------------------------------------------
1145             0000 042A CONT_12         EQU *                     -- connect/disconnect --
1146    0000042A 4A00              TST.B   D0                      check for d0=0
1147    0000042C 6700 03EC         BEQ     DISCONNECT              disconnect will do return
1148    00000430 B03C 0001         CMP.B   #1,D0
1149    00000434 6700 03C0         BEQ     CONNECT                 connect will return
1150             0000 0438 VAL_ERR         EQU *
```

```
00000438 7017              MOVEQ   #IO_MISC,D0             illegal value for register
0000043A 6000 038E         BRA     IOESCAPE
                        *-------------------------------------------------------------------------------
            0000 043E CONT_13       EQU *                   -- hardware handshake --
0000043E 4A00              TST.B   D0                      check for too small value
00000440 6DF6              BLT.S   VAL_ERR                 (located in cont_12)
00000442 B03C 0001         CMP.B   #1,D0                   check for too large value
00000446 6EF0              BGT.S   VAL_ERR
00000448 1540 003B         MOVE.B  D0,MODEM_ON(A2)         assign modem flag
0000044C 4E75              RTS
                        *-------------------------------------------------------------------------------
            0000 044E CONT_14       EQU *                   -- soft reset --
0000044E 4A40              TST.W   D0                      any bit will do reset
00000450 6704              BEQ.S   EXIT_14                 zero value does nothing

00000452 6100 03DE         BSR     SOFT_RESET
00000456 4E75       EXIT_14 RTS
                        *-------------------------------------------------------------------------------
            0000 0458 CONT_15       EQU *                   -- redefine Xon character --
00000458 1540 0041         MOVE.B  D0,XON_CHAR(A2)
0000045C 4E75              RTS
                        *-------------------------------------------------------------------------------
            0000 045E CONT_16       EQU *                   -- redefine Xoff character --
0000045E 1540 0042         MOVE.B  D0,XOFF_CHAR(A2)
00000462 4E75              RTS
                        *-------------------------------------------------------------------------------
            0000 0464 CONT_17       EQU *                   -- redefine ENQ character --
00000464 1540 0043         MOVE.B  D0,ENQ_CHAR(A2)
00000468 4E75              RTS
                        *-------------------------------------------------------------------------------
            0000 046A CONT_18       EQU *                   -- redefine ACK character --
0000046A 1540 0044         MOVE.B  D0,ACK_CHAR(A2)
0000046E 4E75              RTS
                        *-------------------------------------------------------------------------------
            0000 0470 CONT_19       EQU *                   -- redefine FE/PE convert character --
00000470 1540 0045         MOVE.B  D0,CONV_CHAR(A2)
00000474 4E75              RTS
```



```
                        ********************************************************************************
                        *
                        *       interrupt service routine
                        *
                        ********************************************************************************

            0000 0476 RS_RS_ISR     EQU *

                        *
                        *       pascal interface overhead
                        *

00000476 205F              MOVEA.L (SP)+,A0                get return address
00000478 245F              MOVEA.L (SP)+,A2                get temp address
0000047A 226A 0020         MOVEA.L C_ADR(A2),A1            get card address
0000047E 4850              PEA     (A0)                    restore return address

                        *
                        *       verify there is an interrupt
                        *

00000480 1029 0015         MOVE.B  INTR_ID(A1),D0          get interrupt cause
00000484 0800 0000         BTST    #0,D0                   make sure an interrupt is pending
00000488 6702              BEQ.S   INTR_EXIST
0000048A 4E75              RTS                             no interrupt-- return

                        *
                        *       jump to appropriate interrupt handler
                        *

            0000 048C INTR_EXIST    EQU *
0000048C 157C 0001         MOVE.B  #1,IN_ISR(A2)           mark isr processing
         0038
00000492 4880              EXT.W   D0
00000494 303B 0006         MOVE.W  INTR_TABLE(D0),D0       (get appropriate address)
00000498 4EFB 0002         JMP     INTR_TABLE(D0)          (jump to the proper case)

            0000 049C INTR_TABLE    EQU *
0000049C 001A              DC.W    MODEM_INTR-INTR_TABLE       modem change interrupt
0000049E 0038              DC.W    OUTPUT_INTR-INTR_TABLE      output empty interrupt
000004A0 00D0              DC.W    INPUT_INTR-INTR_TABLE       input available interrupt
000004A2 0008              DC.W    ERROR_INTR-INTR_TABLE       error interrupt
```

```
1232                        *
1233                        *   Check for possible unexpected interrupts, if found clear the interrupt
1234                        *      (different for each type of interrupt) and disable that type of
1235                        *      interrupt.
1236
1237              0000 04A4 ERROR_INTR      EQU *
1238    000004A4 1029 001B          MOVE.B  LINE_STAT(A1),D0        clear the interrupt (by reading line status)
1239    000004A8 812A 003F          OR.B    D0,S_LINE(A2)           save line status for user
1240    000004AC 08A9 0002          BCLR    #2,INTR_EN(A1)          disable interrupts since it should not happen
                 0013
1241    000004B2 6000 01BA          BRA     END_ISR
1242
1243              0000 04B6 MODEM_INTR      EQU *
1244    000004B6 4A2A 003B          TST.B   MODEM_ON(A2)            modem handshake on?
1245    000004BA 6706               BEQ.S   ABORT_MODEM               -- no abort
1246    000004BC 2E2A 0028          MOVE.L  BUFO_OFF(A2),D7         output transfer active?
1247    000004C0 6622               BNE.S   XFER_OUT                  -- yes do transfer
1248
1249              0000 04C2 ABORT_MODEM     EQU *
1250    000004C2 1029 001D          MOVE.B  MODEM_STAT(A1),D0       clear the interrupt (by reading modem status)
1251    000004C6 812A 003E          OR.B    D0,S_MODEM(A2)          save modem status for user
1252    000004CA 08A9 0003          BCLR    #3,INTR_EN(A1)          disable interrupt so it won't happen again
                 0013
1253    000004D0 6000 019C          BRA     END_ISR
1254
1255              0000 04D4 OUTPUT_INTR     EQU *
1256    000004D4 2E2A 0028          MOVE.L  BUFO_OFF(A2),D7         output interrupt transfer active ?
1257    000004D8 660A               BNE.S   XFER_OUT                  -- yes do transfer
1258
1259    000004DA 08A9 0001          BCLR    #1,INTR_EN(A1)            -- no disable interrupt
                 0013
1260    000004E0 6000 018C          BRA     END_ISR
```



```
1262                        *
1263                        *   Do the output interrupt transfer
1264                        *       (but only if the THRE and the modem lines are high)
1265                        *
1266
1267              0000 04E4 XFER_OUT        EQU *
1268    000004E4 2647               MOVEA.L D7,A3                   a3 := buffer control block
1269    000004E6 1E29 001B          MOVE.B  LINE_STAT(A1),D7        check for THRE
1270    000004EA 8F2A 003F          OR.B    D7,S_LINE(A2)           save line status for user
1271    000004EE 0807 0005          BTST    #5,D7
1272    000004F2 6700 017A          BEQ     END_ISR
1273
1274    000004F6 4A2A 003B          TST.B   MODEM_ON(A2)
1275    000004FA 6708               BEQ.S   MOVE_OUT
1276    000004FC 6100 04C0          BSR     CHECK_DSR_CTS
1277    00000500 6600 016C          BNE     END_ISR
1278
1279              0000 0504 MOVE_OUT        EQU *
1280    00000504 4247               CLR.W   D7                      clear d7.w
1281    00000506 206B 001C          MOVEA.L TEMP_OFF(A3),A0         a0 := buffer empty pointer
1282    0000050A 1E18               MOVE.B  (A0)+,D7                d7 := character
1283    0000050C 1347 0011          MOVE.B  D7,DATA(A1)             write the character
1284    00000510 2748 001C          MOVE.L  A0,TEMP_OFF(A3)         update empty pointer
1285
1286    00000514 53AB 0010          SUBQ.L  #1,TCNT_OFF(A3)         decrement the count
1287    00000518 6F08               BLE.S   END_XOUT                count=0, transfer ends
1288    0000051A BE6B 000E          CMP.W   TCHR_OFF(A3),D7         character = term. char ?
1289    0000051E 6600 014E          BNE     END_ISR
1290
1291              0000 0522 END_XOUT        EQU *
1292    00000522 4A2B 000B          TST.B   TEND_OFF(A3)            end condition enabled ?
1293    00000526 6730               BEQ.S   CLR_XOUT
1294    00000528 4A2A 003B          TST.B   MODEM_ON(A2)            modem handshake on ?
1295    0000052C 672A               BEQ.S   CLR_XOUT
1296    0000052E 0229 00F5          ANDI.B  #$F5,INTR_EN(A1)        disable output and modem interrupts
                 0013
1297              0000 0534 LOOP_LAST       EQU *
1298    00000534 1E29 001B          MOVE.B  LINE_STAT(A1),D7        wait for everything transferred
1299    00000538 8F2A 003F          OR.B    D7,S_LINE(A2)           before dropping RTS
1300    0000053C 4607               NOT.B   D7
1301    0000053E CE3C 0060          AND.B   #$60,D7
1302    00000542 66F0               BNE.S   LOOP_LAST
1303    00000544 08A9 0001          BCLR    #1,MODEM_CONT(A1)       drop RTS is the EOI condition
                 0019
1304
1305    0000054A 6100 0234          BSR     CLEAR_XFER              clear the transfer
1306    0000054E 4EB9 0000          JSR     LOGEOT                  call the eot procedure
                 0000
1307    00000554 6000 0118          BRA     END_ISR
1308
1309              0000 0558 CLR_XOUT        EQU *
1310    00000558 0229 00F5          ANDI.B  #$F5,INTR_EN(A1)        disable output and modem interrupts
                 0013
1311    0000055E 6100 0220          BSR     CLEAR_XFER              clear the transfer
1312    00000562 4EB9 0000          JSR     LOGEOT                  call the eot procedure
                 0000
1313    00000568 6000 0104          BRA     END_ISR
```

```
1315                        *
1316                        *   Input interrupts are normally active in order to fill the internal
1317                        *       buffer
1318                        *
1319            0000 056C INPUT_INTR      EQU *
1320    0000056C 1229 001B          MOVE.B  LINE_STAT(A1),D1
1321    00000570 1429 0011          MOVE.B  DATA(A1),D2             get the input byte (clears interrupt)
1322    00000574 832A 003F          OR.B    D1,S_LINE(A2)           preserve line status for user
1323
1324    00000578 4A2A 003B          TST.B   MODEM_ON(A2)
1325    0000057C 6712               BEQ.S   CHECK_BREAK             skip modem stuff if handshake off
1326                        *
1327                        *   check for both CD and DSR
1328                        *
1329    0000057E 1029 001D          MOVE.B  MODEM_STAT(A1),D0
1330    00000582 812A 003E          OR.B    D0,S_MODEM(A2)          preserve modem status for user
1331    00000586 4640               NOT     D0
1332    00000588 0240 00A0          ANDI    #$A0,D0                 mask appropriate bits
1333    0000058C 6600 00C4          BNE     INPUT_END               if zero then they were set previously
1334                        *
1335                        *   ignore character if break received
1336                        *
1337            0000 0590 CHECK_BREAK     EQU *
1338    00000590 0801 0004          BTST    #4,D1                   if break received,
1339    00000594 6600 00BC          BNE     INPUT_END               then ignore character
1340                        *
1341                        *   convert Framing and Parity errors to specified character
1342                        *
1343    00000598 1001               MOVE.B  D1,D0                   save line status for later use
1344    0000059A 0240 000C          ANDI    #$0C,D0                 check for PARITY and FRAMING errors
1345    0000059E 6704               BEQ.S   NO_CONVERT
1346    000005A0 142A 0045          MOVE.B  CONV_CHAR(A2),D2        convert the character
1347
1348            0000 05A4 NO_CONVERT      EQU *
1349                        *
1350                        *       jump to appropriate handshake handler
1351                        *
1352    000005A4 102A 0040          MOVE.B  S_HANDSH(A2),D0
1353    000005A8 4880               EXT.W   D0
1354    000005AA 303B 0006          MOVE.W  HAND_TABLE(D0),D0       get address to jump
1355    000005AE 4EFB 0002          JMP     HAND_TABLE(D0)
1356
1357            0000 05B2 HAND_TABLE      EQU *
1358    000005B2 0068               DC.W    ENQ_HAND-HAND_TABLE
1359    000005B4 0008               DC.W    XON_HAND-HAND_TABLE
1360    000005B6 0096               DC.W    NO_HAND-HAND_TABLE
1361    000005B8 0096               DC.W    NO_HAND-HAND_TABLE
```



```
1363            0000 05BA XON_HAND        EQU *
1364                        *
1365                        *   Do host part of the handshake -- check for Xon and Xoff
1366                        *
1367    000005BA B42A 0041          CMP.B   XON_CHAR(A2),D2
1368    000005BE 6626               BNE.S   CHECK_XOFF
1369    000005C0 157C 0001          MOVE.B  #1,XMITTING(A2)         turn transmitting back on
                003D
1370
1371    000005C6 4AAA 0028          TST.L   BUFO_OFF(A2)            \
1372    000005CA 6700 0086          BEQ     INPUT_END               \
1373    000005CE 08E9 0001          BSET    #1,INTR_EN(A1)            \  enable interrupts if output
                0013
1374    000005D4 4A2A 003B          TST.B   MODEM_ON(A2)              /  transfer is active
1375    000005D8 6700 0078          BEQ     INPUT_END               /
1376    000005DC 08E9 0003          BSET    #3,INTR_EN(A1)          /
                0013
1377    000005E2 6000 006E          BRA     INPUT_END
1378
1379            0000 05E6 CHECK_XOFF      EQU *
1380    000005E6 B42A 0042          CMP.B   XOFF_CHAR(A2),D2
1381    000005EA 660E               BNE.S   TERM_HAND
1382    000005EC 422A 003D          CLR.B   XMITTING(A2)            turn transmitting off
1383
1384    000005F0 137C 0001          MOVE.B  #$1,INTR_EN(A1)         turn any possible output interrupts off
                0013
1385    000005F6 6000 005A          BRA     INPUT_END
1386
1387                        *
1388                        *   Do terminal part of the handshake
1389                        *
1390            0000 05FA TERM_HAND       EQU *
1391    000005FA 4A2A 003C          TST.B   RECEIVING(A2)           if receiving is on, might have
1392    000005FE 6748               BEQ.S   PUTINQ                    to turn it off
1393    00000600 6100 043C          BSR     QUEUE_SPACE             d3 := space left in queue
1394    00000604 B67C 0028          CMP.W   #XOFF_SIZE,D3
1395    00000608 6C3E               BGE.S   PUTINQ
1396                        *
1397                        *   Have to turn receiving off
1398                        *
1399    0000060A 162A 0042          MOVE.B  XOFF_CHAR(A2),D3        prepare to send Xoff
1400    0000060E 6100 0320          BSR     SEND                    send character
1401    00000612 6634               BNE.S   PUTINQ                  send did not succeed
1402    00000614 422A 003C          CLR.B   RECEIVING(A2)           no longer expecting input
1403    00000618 602E               BRA.S   PUTINQ                  but put present char in queue
1404
```

```
1406
1407            0000 061A ENQ_HAND        EQU *
1408    0000061A B42A 0043        CMP.B   ENQ_CHAR(A2),D2         IF char <> ENQ
1409    0000061E 6600 0028        BNE     PUTINQ                    THEN put char in queue
1410    00000622 6100 041A        BSR     QUEUE_SPACE               ELSE
1411    00000626 B67C 0050        CMP.W   #ACK_SIZE,D3                IF queue_space <= 80
1412    0000062A 6C08             BGE.S   SEND_ACK
1413    0000062C 422A 003C        CLR.B   RECEIVING(A2)                   receiving := false
1414    00000630 6000 0020        BRA     INPUT_END
1415
1416            0000 0634 SEND_ACK        EQU *
1417    00000634 162A 0044        MOVE.B  ACK_CHAR(A2),D3         set up parameter in D3
1418    00000638 6100 02F6        BSR     SEND                    send ACK
1419    0000063C 6700 0014        BEQ     INPUT_END
1420
1421    00000640 422A 003C        CLR.B   RECEIVING(A2)           not receiving since ACK not sent
1422    00000644 6000 000C        BRA     INPUT_END
1423
1424                      *
1425                      *   Put character (d2) in queue, and check for overrun.
1426                      *
1427            0000 0648 NO_HAND         EQU *
1428            0000 0648 PUTINQ          EQU *
1429    00000648 6100 03A0        BSR     QUEUE_FULL      IF queue_full
1430    0000064C 670A             BEQ.S   OVERRUN            THEN overrun_error
1431    0000064E 6100 03B6        BSR     INQUEUE            ELSE inqueue(char)
1432
1433            0000 0652 INPUT_END       EQU *
1434    00000652 0241 0002        ANDI    #$02,D1         check for overrun error
1435    00000656 6708             BEQ.S   CHECK_XIN
1436
1437            0000 0658 OVERRUN         EQU *
1438    00000658 257C 0000        MOVE.L  #OVERRUN_ERROR,S_ERROR(A2)
             013A 0034
1439
1440                      *
1441                      *   If transfer in is active then do the transfer
1442                      *
1443            0000 0660 CHECK_XIN       EQU *
1444    00000660 4A2A 0039        TST.B   XIN_ACT(A2)             check for transfer active
1445    00000664 6708             BEQ.S   END_ISR
1446
1447    00000666 266A 0024        MOVEA.L BUFI_OFF(A2),A3         a3 := buffer control block pointer
1448    0000066A 6100 00D8        BSR     DUMP_BUFFER
1449
1450            0000 066E END_ISR         EQU *
1451    0000066E 422A 0038        CLR.B   IN_ISR(A2)              not in isr any longer
1452    00000672 4E75             RTS
```



```
1455                      ********************************************************************************
1456                      *
1457                      *       transfer
1458                      *
1459                      ********************************************************************************
1460
1461            0000 0674 RS_RS_TFR       EQU *
1462
1463                      *
1464                      *   Pascal interface overhead
1465                      *
1466    00000674 205F             MOVEA.L (SP)+,A0                get return address
1467    00000676 265F             MOVEA.L (SP)+,A3                get buffer control block address
1468    00000678 245F             MOVEA.L (SP)+,A2                get temp address
1469    0000067A 226A 0020        MOVEA.L C_ADR(A2),A1            get card address
1470    0000067E 4850             PEA     (A0)                    restore return address
1471                      *
1472                      *   Card overhead
1473                      *
1474    00000680 6100 0174        BSR     CONNECT
1475    00000684 6100 01EE        BSR     CHECK_ERROR
1476                      *
1477                      *   Check for unsupported transfer modes
1478                      *       ( done by table jump also )
1479                      *
1480    00000688 4A2B 000A        TST.B   T_BW_OFF(A3)            word mode?
1481    0000068C 662E             BNE.S   WORD_ERR                  -- is unsupported
1482
1483    0000068E 122B 0009        MOVE.B  TUSR_OFF(A3),D1         d1.w := offset into transfer table
1484    00000692 4881             EXT.W   D1
1485    00000694 D241             ADD.W   D1,D1                   d1.w := word offset into table
1486    00000696 323B 1006        MOVE.W  XFER_TABLE(D1),D1
1487    0000069A 4EFB 1002        JMP     XFER_TABLE(D1)
1488
1489            0000 069E XFER_TABLE      EQU *
1490    0000069E 0014             DC.W    XFER_ERR-XFER_TABLE     not used
1491    000006A0 0014             DC.W    DMA_ERR-XFER_TABLE      serial DMA -- not supported
1492    000006A2 0028             DC.W    SER_FHS-XFER_TABLE      serial FHS
1493    000006A4 0028             DC.W    SER_FHS-XFER_TABLE      serial fastest -- same as serial FHS
1494    000006A6 0014             DC.W    XFER_ERR-XFER_TABLE     not used
1495
1496    000006A8 0036             DC.W    INTR_XFER-XFER_TABLE    overlap INTR
1497    000006AA 0014             DC.W    DMA_ERR-XFER_TABLE      overlap DMA -- not supported
1498    000006AC 0014             DC.W    XFER_ERR-XFER_TABLE     overlap FHS -- not supported
1499    000006AE 0036             DC.W    INTR_XFER-XFER_TABLE    overlap FASTEST -- same as overlap INTR
1500    000006B0 0036             DC.W    INTR_XFER-XFER_TABLE    overlap overlap -- same as overlap INTR
```

```
1502                            *
1503                            *   Error escapes
1504                            *
1505                0000 06B2 END_ERR           EQU *
1506                0000 06B2 DMA_ERR           EQU *
1507                0000 06B2 XFER_ERR          EQU *
1508    000006B2 6100 00CC              BSR     CLEAR_XFER
1509    000006B6 7007                   MOVEQ   #TFR_ERR,D0
1510    000006B8 6000 0110              BRA     IOESCAPE
1511
1512                0000 06BC WORD_ERR          EQU *
1513
1514    000006BC 6100 00C2              BSR     CLEAR_XFER
1515    000006C0 700E                   MOVEQ   #NO_WORD,D0
1516    000006C2 6000 0106              BRA     IOESCAPE
1517                            *
1518                            *   Set the actual mode for transfers
1519                            *
1520                0000 06C6 SER_FHS           EQU *
1521    000006C6 177C 0004              MOVE.B  #TT_FHS,TACT_OFF(A3)    set the actual mode
                    0007
1522    000006CC 4A2B 000D              TST.B   TDIR_OFF(A3)            jump to correct direction handler
1523    000006D0 6634                   BNE.S   OUTPUT_XFER
1524    000006D2 600C                   BRA.S   INPUT_XFER
1525
1526                0000 06D4 INTR_XFER         EQU *
1527    000006D4 177C 0001              MOVE.B  #TT_INT,TACT_OFF(A3)    set actual mode to INTR
                    0007
1528    000006DA 4A2B 000D              TST.B   TDIR_OFF(A3)            jump to correct direction handler
1529    000006DE 6626                   BNE.S   OUTPUT_XFER
1530                            *       BRA.S   INPUT_XFER
1531
1532                            *
1533                            *   Input transfer setup.
1534                            *
1535
1536                0000 06E0 INPUT_XFER        EQU *
1537    000006E0 4A2B 000B              TST.B   TEND_OFF(A3)            end condition not allowed on input xfers
1538    000006E4 66CC                   BNE.S   END_ERR
1539
1540    000006E6 6100 005C              BSR     DUMP_BUFFER             do most of transfers with intr enabled
1541    000006EA 6756                   BEQ.S   EXIT_TFR                if transfer done, then exit
1542
1543    000006EC 4229 0003              CLR.B   INTR_SW(A1)             disable interrupts for critical section
1544    000006F0 157C 0001              MOVE.B  #1,XIN_ACT(A2)
                    0039
1545    000006F6 6100 00A4              BSR     SET_XFER                set interface busy
1546    000006FA 6100 0048              BSR     DUMP_BUFFER             make sure that buffer is empty (prevent deadlock)
1547                            *                                       if eot the following code will exit
1548                            *                                       so no explicit exit is done
1549    000006FE 08E9 0007              BSET    #7,INTR_SW(A1)          end of critical section
                    0003
1550
1551    00000704 6026                   BRA.S   CHECK_FHS               end of input transfer setup
```



```
1553                            *
1554                            *   Output transfer setup
1555                            *
1556
1557                0000 0706 OUTPUT_XFER       EQU *
1558
1559    00000706 4229 0003              CLR.B   INTR_SW(A1)             disable interrupts for critical section
1560
1561    0000070A 6100 0090              BSR     SET_XFER                set interface busy
1562    0000070E 08E9 0001              BSET    #1,MODEM_CONT(A1)       set RTS
                    0019
1563    00000714 4A2A 003D              TST.B   XMITTING(A2)            if not transmitting, don't enable interrupts
1564    00000718 6712                   BEQ.S   CHECK_FHS
1565    0000071A 08E9 0001              BSET    #1,INTR_EN(A1)          enable output interrupts
                    0013
1566    00000720 4A2A 003B              TST.B   MODEM_ON(A2)            if modem handshake
1567    00000724 6706                   BEQ.S   CHECK_FHS
1568    00000726 08E9 0003              BSET    #3,INTR_EN(A1)          enable modem interrupts
                    0013
1569
1570                            *
1571                            *   IF serial transfer THEN wait until transfer is done
1572                            *
1573                0000 072C CHECK_FHS         EQU *
1574    0000072C 08E9 0007              BSET    #7,INTR_SW(A1)          end of critical section
                    0003
1575
1576    00000732 0C2B 0004              CMPI.B  #TT_FHS,TACT_OFF(A3)
                    0007
1577    00000738 6608                   BNE.S   EXIT_TFR
1578                0000 073A WAIT_FHS          EQU *
1579    0000073A 0C2B 00FF              CMPI.B  #255,T_SC_OFF(A3)       wait until buffer is not busy
                    0005
1580    00000740 66F8                   BNE.S   WAIT_FHS
1581                0000 0742 EXIT_TFR          EQU *
1582    00000742 4E75                   RTS
```

```
1585                        ********************************************************************************
1586                        *
1587                        *       TRANSFER SUPPORT ROUTINES
1588                        *
1589                        ********************************************************************************
1590                        *
1591                        *       DUMP BUFFER
1592                        *          transfer from the internal queue to user queue
1593                        *
1594                        *       ON ENTRY: a3 - points to buffer control block
1595                        *       ON EXIT : IF transfer was completed
1596                        *                    THEN d2.L=0 and Z=1
1597                        *                    ELSE d2.L=1 and Z=0
1598                        *       USES:     a0 - current fill pointer to user input buffer
1599                        *                 d2 - character being transfered
1600                        *-------------------------------------------------------------------------------
1601
1602             0000 0744 DUMP_BUFFER     EQU *
1603    00000744 206B 0020          MOVEA.L TFIL_OFF(A3),A0         get fill pointer
1604    00000748 4242               CLR.W   D2                      clear top half of D2 (for later compares)
1605
1606             0000 074A DUMP_LOOP       EQU *
1607    0000074A 6100 0294          BSR     QUEUE_EMPTY
1608    0000074E 6728               BEQ.S   EXIT_DUMP
1609    00000750 6100 0162          BSR     GET_CHAR                d2 := character
1610    00000754 10C2               MOVE.B  D2,(A0)+                put it in the linear buffer
1611    00000756 53AB 0010          SUBQ.L  #1,TCNT_OFF(A3)         decrement count
1612    0000075A 6F06               BLE.S   END_XIN
1613    0000075C B46B 000E          CMP.W   TCHR_OFF(A3),D2
1614    00000760 66E8               BNE.S   DUMP_LOOP
1615
1616             0000 0762 END_XIN         EQU *
1617    00000762 2748 0020          MOVE.L  A0,TFIL_OFF(A3)         update fill pointer
1618    00000766 422A 0039          CLR.B   XIN_ACT(A2)
1619    0000076A 6100 0014          BSR     CLEAR_XFER
1620    0000076E 4EB9 0000          JSR     LOGEOT
                0000
1621
1622    00000774 4282               CLR.L   D2                      set Z flag to mark transfer ended
1623    00000776 4E75               RTS
1624
1625             0000 0778 EXIT_DUMP       EQU *
1626    00000778 2748 0020          MOVE.L  A0,TFIL_OFF(A3)         update fill pointer
1627
1628    0000077C 7401               MOVEQ   #1,D2                   clear Z flage to mark transfer still active
1629    0000077E 4E75               RTS
```



```
1631                        *-------------------------------------------------------------------------------
1632                        *       CLEAR_XFER
1633                        *          make a transfer inactive (unlink temp space and buffer control block)
1634                        *       ON ENTRY:  a3 - points to the buffer control block
1635                        *-------------------------------------------------------------------------------
1636
1637             0000 0780 CLEAR_XFER      EQU *
1638    00000780 422B 0007          CLR.B   TACT_OFF(A3)            clear actual transfer mode
1639    00000784 177C 00FF          MOVE.B  #255,T_SC_OFF(A3)       set the buffer not busy
                0005
1640    0000078A 4A2B 000D          TST.B   TDIR_OFF(A3)
1641    0000078E 6606               BNE.S   CLEAR_OUT
1642    00000790 42AA 0024          CLR.L   BUFI_OFF(A2)            clear input transfer
1643    00000794 4E75               RTS
1644             0000 0796 CLEAR_OUT       EQU *
1645    00000796 42AA 0028          CLR.L   BUFO_OFF(A2)            clear output transfer
1646    0000079A 4E75               RTS
1647
1648                        *-------------------------------------------------------------------------------
1649                        *       SET_XFER
1650                        *          make a transfer active (link temp space with buffer control block)
1651                        *       ON ENTRY:  a3 - the buffer control block
1652                        *-------------------------------------------------------------------------------
1653
1654             0000 079C SET_XFER        EQU *
1655    0000079C 176A 002D          MOVE.B  IO_SC(A2),T_SC_OFF(A3)  set the buffer busy
                0005
1656    000007A2 4A2B 000D          TST.B   TDIR_OFF(A3)
1657    000007A6 6606               BNE.S   SET_OUT
1658    000007A8 254B 0024          MOVE.L  A3,BUFI_OFF(A2)         set sc's input active
1659    000007AC 4E75               RTS
1660             0000 07AE SET_OUT         EQU *
1661    000007AE 254B 0028          MOVE.L  A3,BUFO_OFF(A2)         set sc's output active
1662    000007B2 4E75               RTS
1663
1664                        *-------------------------------------------------------------------------------
1665                        *       CHECK_XFER_IN, CHECK_XFER_OUT
1666                        *          gives an error if a transfer is active
1667                        *       USES:  d0 -- only if an ioescape is to be given
1668                        *-------------------------------------------------------------------------------
1669
1670             0000 07B4 CHECK_XFER_IN   EQU *
1671    000007B4 4AAA 0024          TST.L   BUFI_OFF(A2)
1672    000007B8 660A               BNE.S   BUSY_ERR
1673    000007BA 4E75               RTS
1674
1675             0000 07BC CHECK_XFER_OUT  EQU *
1676    000007BC 4AAA 0028          TST.L   BUFO_OFF(A2)
1677    000007C0 6602               BNE.S   BUSY_ERR
1678    000007C2 4E75               RTS
1679
1680             0000 07C4 BUSY_ERR        EQU *
1681    000007C4 7008               MOVEQ   #SC_BUSY,D0
1682    000007C6 6000 0002          BRA     IOESCAPE
```

```
1685                        ********************************************************************************
1686                        *
1687                        *       Useful Subroutines
1688                        *
1689                        ********************************************************************************
1690                        *
1691                        *       IOESCAPE
1692                        *          ON ENTRY:  d0.L -- contains the escape code
1693                        *--------------------------------------------------------------------------------
1694
1695             0000 07CA IOESCAPE        EQU *
1696    000007CA 2B40 FFBE         MOVE.L  D0,IOE_RSLT(A5)         *  put ioe_result
1697    000007CE 4280              CLR.L   D0                      *<<< BUG FIX >>>
1698    000007D0 102A 002D         MOVE.B  IO_SC(A2),D0            *  get select code of card
1699    000007D4 2B40 FFBA         MOVE.L  D0,IOE_SC(A5)           *  put ioe_sc
1700    000007D8 3B7C FFE6         MOVE.W  #IOE_ERROR,ESC_CODE(A5) *  escapecode := ioe_error
                 FFFE
1701    000007DE 4E4A              TRAP    #10                     *  do Pascal escape
1702
1703                        *--------------------------------------------------------------------------------
1704                        *       RDIVU
1705                        *          unsigned integer div1de rounded.
1706                        *          ON ENTRY:  d0.w -- divisor (unchanged by this routine)
1707                        *                     d1.l -- dividend
1708                        *          ON EXIT:   d1.w -- rounded quotient
1709                        *--------------------------------------------------------------------------------
1710             0000 07E0 RDIVU           EQU *
1711    000007E0 82C0              DIVU    D0,D1                   do truncated division
1712    000007E2 4841              SWAP    D1                      get access to remainder
1713    000007E4 E349              LSL.W   #1,D1                   multiply remainder by 2
1714    000007E6 6508              BCS.S   ROUND                   if carry then remainder*2>divisor
1715    000007E8 B041              CMP.W   D1,D0                   remainder*2 > divisor ?
1716    000007EA 6F04              BLE.S   ROUND                   round up if so.
1717                        *                                       --do not round --
1718    000007EC 4841              SWAP    D1                      get quotient
1719    000007EE 4E75              RTS
1720             0000 07F0 ROUND           EQU *                   --round up--
1721    000007F0 4841              SWAP    D1                      get old quotient
1722    000007F2 5241              ADDQ.W  #1,D1                   increment (do the rounding)
1723    000007F4 4E75              RTS
```



```
1725                        *--------------------------------------------------------------------------------
1726                        *       CONNECT
1727                        *          connects the card if not connected already.
1728                        *
1729                        *       uses : d6,d7 by called routines
1730                        *--------------------------------------------------------------------------------
1731
1732             0000 07F6 CONNECT         EQU *
1733    000007F6 4A2A 003A         TST.B   CONNECTED(A2)           IF connected THEN do nothing
1734    000007FA 661C              BNE.S   EXIT_CONNECT
1735
1736    000007FC 08E9 0000         BSET    #0,MODEM_CONT(A1)       set DTR
                 0019
1737    00000802 6100 002E         BSR     SOFT_RESET              initialize the dynamic data
1738    00000806 137C 0001         MOVE.B  #1,INTR_EN(A1)          enable receive interrupts
                 0013
1739    0000080C 157C 0001         MOVE.B  #1,CONNECTED(A2)        set connected
                 003A
1740    00000812 08E9 0007         BSET    #7,INTR_SW(A1)          enable card interrupts
                 0003
1741             0000 0818 EXIT_CONNECT    EQU *
1742    00000818 4E75              RTS
1743
1744
1745                        *--------------------------------------------------------------------------------
1746                        *       DISCONNECT
1747                        *          disconnect and disable interrupts
1748                        *--------------------------------------------------------------------------------
1749
1750             0000 081A DISCONNECT      EQU *
1751    0000081A 08A9 0007         BCLR    #7,INTR_SW(A1)          disable card interrupts
                 0003
1752    00000820 422A 003A         CLR.B   CONNECTED(A2)           set disconnected
1753    00000824 0229 00FC         ANDI.B  #$FC,MODEM_CONT(A1)     drop DTR and RTS
                 0019
1754    0000082A 4E71              NOP
1755    0000082C 4229 0013         CLR.B   INTR_EN(A1)             disable all UART interrupts
1756    00000830 4E75              RTS
```

```
1758                        *-------------------------------------------------------------------------------
1759                        *       SOFT_RESET
1760                        *          initialize the "dynamic" attributes of the drivers
1761                        *
1762                        *       uses :  d6,d7 as temporary
1763                        *-------------------------------------------------------------------------------
1764
1765             0000 0832 SOFT_RESET      EQU *
1766
1767    00000832 4EB9 0000          JSR     ABORT_IO                abort transfers
                 0000
1768
1769    00000838 1C29 0003          MOVE.B  INTR_SW(A1),D6          save interrupt state
1770    0000083C 4229 0003          CLR.B   INTR_SW(A1)             disable interrupts
1771
1772    00000840 0229 0001          ANDI.B  #1,INTR_EN(A1)          disable modem and transmit interrupts
                 0013
1773    00000846 6100 0188          BSR     INIT_QUEUE
1774    0000084A 1E29 0011          MOVE.B  DATA(A1),D7             destroy any data
1775    0000084E 422A 003F          CLR.B   S_LINE(A2)
1776    00000852 1E29 001B          MOVE.B  LINE_STAT(A1),D7        reset the line status (destructive read)
1777    00000856 422A 003E          CLR.B   S_MODEM(A2)
1778    0000085A 1E29 001D          MOVE.B  MODEM_STAT(A1),D7       reset the modem status (destructive read)
1779
1780
1781    0000085E 42AA 0034          CLR.L   S_ERROR(A2)
1782    00000862 157C 0001          MOVE.B  #1,RECEIVING(A2)
                 003C
1783    00000868 157C 0001          MOVE.B  #1,XMITTING(A2)
                 003D
1784
1785    0000086E 1346 0003          MOVE.B  D6,INTR_SW(A1)          restore the interrupt state
1786    00000872 4E75               RTS
1787
1788
1789                        *-------------------------------------------------------------------------
1790                        *       CHECK_ERROR
1791                        *          check for errors recorded in interrupt service routines (ISRs)
1792                        *          USES:  D0,D7 only if doing ioescape
1793                        *-------------------------------------------------------------------------
1794
1795             0000 0874 CHECK_ERROR     EQU *
1796    00000874 4AAA 0034          TST.L   S_ERROR(A2)             is error present
1797    00000878 6602               BNE.S   ERROR_EXIST
1798    0000087A 4E75               RTS                             return if not error
1799             0000 087C ERROR_EXIST     EQU *
1800    0000087C 1E29 0003          MOVE.B  INTR_SW(A1),D7          save interrupt condition
1801    00000880 4229 0003          CLR.B   INTR_SW(A1)             disable interrupt for critical section
1802
1803    00000884 202A 0034          MOVE.L  S_ERROR(A2),D0          get error
1804    00000888 42AA 0034          CLR.L   S_ERROR(A2)             clear errors
1805
1806    0000088C 1347 0003          MOVE.B  D7,INTR_SW(A1)          restore interrupts
1807    00000890 6000 FF38          BRA     IOESCAPE                do pascal escape
```



```
1809                        *-------------------------------------------------------------------------
1810                        *       WAIT_SEND
1811                        *          This routine waits for the transmitting flag then sends
1812                        *          a character.  It escapes if SEND returns with an error.
1813                        *       NOTE:  this routine cannot be called by ISRs!!!
1814                        *       ON ENTRY:  d3.B -- character to be sent
1815                        *       USES    :  a4 -- used by called routines
1816                        *                  d4,d6,d7 -- by called routines
1817                        *-------------------------------------------------------------------------
1818
1819             0000 0894 WAIT_SEND       EQU *
1820
1821                        *
1822                        *   Wait for xmitting flag (no timeouts !!)
1823                        *       (the wait is important for Xon/Xoff as host)
1824                        *
1825    00000894 4A2A 003D          TST.B   XMITTING(A2)
1826    00000898 67FA               BEQ.S   WAIT_SEND
1827                        *
1828                        *   Send the character
1829                        *
1830             0000 089A OK_XMIT         EQU *
1831    0000089A 6100 0094          BSR     SEND                    send character with timeout
1832    0000089E 61D4               BSR     CHECK_ERROR             check for errors found by send
1833    000008A0 4E75               RTS
1834
1835
1836                        *-------------------------------------------------------------------------
1837                        *       WAIT_GET
1838                        *          wait until the queue is empty before getting a character
1839                        *       ON EXIT:  D2.B contains the character
1840                        *                 (the rest of D2 is not altered!)
1841                        *       USES:     A4.L -- parameter to WAIT
1842                        *                 D0,D3,D4,D6,D7 -- used by called routines
1843                        *-------------------------------------------------------------------------
1844
1845             0000 08A2 WAIT_GET        EQU *
1846
1847                        *
1848                        *   Wait (with timeout) for queue not empty
1849                        *
1850    000008A2 49FA 010C          LEA     CHECK_QUEUE,A4          \  call wait with the not queue empty
1851    000008A6 6100 00CC          BSR     WAIT                    /  function
1852    000008AA 61C8               BSR     CHECK_ERROR             check for wait error
1853
1854    000008AC 6100 0006          BSR     GET_CHAR
1855    000008B0 61C2               BSR     CHECK_ERROR
1856    000008B2 4E75               RTS
```

```
1858                        *-------------------------------------------------------------------------------
1859                        *       GET_CHAR
1860                        *          get a character with software handshake.
1861                        *       ON ENTRY :  the queue is not empty!
1862                        *       ON EXIT:  D2.B contains the character
1863                        *                 (the rest of D2 is not altered!)
1864                        *       USES:     D3 -- space left in queue/temporary for character
1865                        *                 D0.W -- handshake type & temporary
1866                        *                 A4,D4,D6,D7 -- temporary
1867                        *-------------------------------------------------------------------------------
1868
1869
1870             0000 08B4 GET_CHAR        EQU *
1871
1872                        *
1873                        *   Read the character ( and pass it back )
1874                        *
1875     000008B4 6100 016C         BSR     OUTQUEUE                get the character (into D2)
1876                        *
1877                        *   Check for and do handshake overhead
1878                        *
1879     000008B8 4A2A 003C         TST.B   RECEIVING(A2)           if receiving
1880     000008BC 6670              BNE.S   READ_END                then no overhead needed
1881                        *
1882                        *   Jump to appropriate handshake handler
1883                        *
1884     000008BE 102A 0040         MOVE.B  S_HANDSH(A2),D0         get handshake
1885     000008C2 4880              EXT.W   D0
1886     000008C4 303B 0006         MOVE.W  H_TABLE(D0),D0
1887     000008C8 4EFB 0002         JMP     H_TABLE(D0)
1888
1889             0000 08CC H_TABLE         EQU *
1890     000008CC 0008              DC.W    ENQ_H-H_TABLE
1891     000008CE 0018              DC.W    XON_H-H_TABLE
1892     000008D0 0062              DC.W    READ_END-H_TABLE        no handshake (no overhead)
1893     000008D2 0062              DC.W    READ_END-H_TABLE        no handshake
```



```
1895                        *
1896                        *   ENQ/ACK handshake--send ACK if queue can handle more than 80 chars
1897                        *
1898             0000 08D4 ENQ_H           EQU *
1899     000008D4 6100 0168         BSR     QUEUE_SPACE             returns space left in D3
1900     000008D8 B67C 0050         CMP.W   #ACK_SIZE,D3
1901     000008DC 6D50              BLT.S   READ_END                space not big enough to send ACK
1902                        *
1903                        *   Send character to indicate card is receiving and set receiving flag
1904                        *
1905     000008DE 162A 0044         MOVE.B  ACK_CHAR(A2),D3         send ack
1906     000008E2 600E              BRA.S   SEND_HAND
1907
1908                        *
1909                        *   Xon/Xoff handshake--send Xon if queue can handle more characters
1910                        *
1911             0000 08E4 XON_H           EQU *
1912     000008E4 6100 0158         BSR     QUEUE_SPACE             d3 := space left in queue
1913     000008E8 B67C 0060         CMP.W   #XON_SIZE,D3
1914     000008EC 6D40              BLT.S   READ_END                space not big enough to send XON
1915                        *
1916                        *   Send character to indicate card is receiving and set receiving flag
1917                        *
1918     000008EE 162A 0041         MOVE.B  XON_CHAR(A2),D3
1919
1920             0000 08F2 SEND_HAND       EQU *                   send handshake character (in D3)
1921     000008F2 137C 0001         MOVE.B  #1,INTR_EN(A1)          only have receive interrupt enabled
                 0013
1922     000008F8 6100 0036         BSR     SEND                    send char which is in d2
1923     000008FC 6606              BNE.S   RESTORE
1924     000008FE 157C 0001         MOVE.B  #1,RECEIVING(A2)        turn receiving back on
                 003C
1925             0000 0904 RESTORE         EQU *                   recalculate interrupt enable mask
1926     00000904 1E29 0003         MOVE.B  INTR_SW(A1),D7          save interrupt status
1927     00000908 4229 0003         CLR.B   INTR_SW(A1)             critical section
1928     0000090C 4AAA 0028         TST.L   BUFO_OFF(A2)
1929     00000910 6718              BEQ.S   END_RESTORE
1930     00000912 4A2A 003D         TST.B   XMITTING(A2)
1931     00000916 6712              BEQ.S   END_RESTORE
1932     00000918 08E9 0001         BSET    #1,INTR_EN(A1)
                 0013
1933     0000091E 4A2A 003B         TST.B   MODEM_ON(A2)
1934     00000922 6706              BEQ.S   END_RESTORE
1935     00000924 08E9 0003         BSET    #3,INTR_EN(A1)
                 0013
1936             0000 092A END_RESTORE     EQU *
1937     0000092A 1347 0003         MOVE.B  D7,INTR_SW(A1)          end critical section
1938
1939             0000 092E READ_END        EQU *
1940     0000092E 4E75              RTS
```

```
1942                        *-------------------------------------------------------------------------------
1943                        *       SEND
1944                        *       ON ENTRY:  d3.B -- character to be sent
1945                        *       ON EXIT :  IF character sent
1946                        *                     THEN Z=1
1947                        *                     ELSE Z=0, S_ERROR updated to newest error
1948                        *       USES    : a4 -- parameter to WAIT
1949                        *                 d7 -- temporary
1950                        *                 d6 -- by called routines
1951                        *-------------------------------------------------------------------------------
1952
1953                0000 0930 SEND          EQU *
1954    00000930 08E9 0001          BSET    #1,MODEM_CONT(A1)       set RTS
                    0019
1955                        *
1956                        *   Wait (with timeout) for transmit registers empty
1957                        *
1958                0000 0936 LOOP_THRE     EQU *
1959    00000936 1E29 001B          MOVE.B  LINE_STAT(A1),D7
1960    0000093A 8F2A 003F          OR.B    D7,S_LINE(A2)           save line status for user
1961    0000093E CE3C 0020          AND.B   #$20,D7                 look at THRE bit
1962    00000942 67F2               BEQ.S   LOOP_THRE
1963
1964    00000944 4A2A 003B          TST.B   MODEM_ON(A2)            skip modem stuff if modem handshake off
1965    00000948 6722               BEQ.S   XMIT_CHAR
1966                        *
1967                        *   Modem checking depends on if this routine was called from an ISR
1968                        *
1969    0000094A 4A2A 0038          TST.B   IN_ISR(A2)
1970    0000094E 6710               BEQ.S   NOT_ISR
1971
1972    00000950 6100 006C          BSR     CHECK_DSR_CTS
1973    00000954 6716               BEQ.S   XMIT_CHAR               modem lines are up, goto transmit
1974
1975    00000956 257C 0000          MOVE.L  #316,S_ERROR(A2)        CTS false error
                    013C 0034
1976    0000095E 4E75               RTS                             side effect -- Z:=0
1977                        *
1978                        *   Wait (with timeout) for DSR and CTS
1979                        *
1980                0000 0960 NOT_ISR       EQU *
1981    00000960 49FA 005C          LEA     CHECK_DSR_CTS,A4        \  call WAIT with appropriate
1982    00000964 6100 000E          BSR     WAIT                    /  function parameter
1983    00000968 6702               BEQ.S   XMIT_CHAR               no errors, goto transmit
1984    0000096A 4E75               RTS                             (Z=0 still)
1985                        *
1986                        *   Send the character (in d3)
1987                        *
1988                0000 096C XMIT_CHAR     EQU *
1989    0000096C 1343 0011          MOVE.B  D3,DATA(A1)             do actual transmit
1990    00000970 4207               CLR.B   D7                      indicate no errors  (Z := 1)
1991    00000972 4E75               RTS
```



```
1993                        *-------------------------------------------------------------------------------
1994                        *       WAIT
1995                        *          this function waits with timeout for a condition to happen,
1996                        *          if the condition does not happen within the timeout, then
1997                        *          S_ERROR(A2) is marked with the timeout error.
1998                        *       ON ENTRY:  A4.L  points to the routine which will determine if
1999                        *                        the condition is met.  This routine should have
2000                        *                        the following conditions:
2001                        *                          --uses at the most d7,d6
2002                        *                          --returns Z=1 if the condition is met
2003                        *                          --all routines should have similar timing
2004                        *       ON EXIT:   IF error is found
2005                        *                     THEN Z=0, S_ERROR indicates the error
2006                        *                     ELSE Z=1
2007                        *       USES    :  D4.L  --  timeout counter
2008                        *                  D7,D6 --  can be used by called routine (see above)
2009                        *-------------------------------------------------------------------------------
2010
2011                0000 0974 WAIT          EQU *
2012    00000974 4E94               JSR     (A4)                    check the condition
2013    00000976 6736               BEQ.S   EXIT_WAIT               exit if condition met (Z=1)
2014
2015    00000978 282A 002E          MOVE.L  TIMEOUT(A2),D4
2016    0000097C 6726               BEQ.S   WAIT_LOOP2              infinite timeout if value is 0.
2017
2018    0000097E 2E04               MOVE.L  D4,D7                   \
2019    00000980 E38F               LSL.L   #1,D7                    \   initialize counter
2020    00000982 E58C               LSL.L   #2,D4                     \     (multiply by 54)
2021    00000984 D887               ADD.L   D7,D4                      |
2022    00000986 2E04               MOVE.L  D4,D7                      /
2023    00000988 E78C               LSL.L   #3,D4                     /
2024    0000098A D887               ADD.L   D7,D4                    /
2025
2026                0000 098C WAIT_LOOP     EQU *
2027    0000098C 4AAA 0034          TST.L   S_ERROR(A2)             check for errors saved during wait
2028    00000990 661C               BNE.S   EXIT_WAIT                  exit (Z=0)
2029    00000992 4E94               JSR     (A4)                       check the condition
2030    00000994 6718               BEQ.S   EXIT_WAIT                  exit if conditon met (Z=1)
2031
2032    00000996 5384               SUBQ.L  #1,D4                      counter := counter - 1
2033    00000998 6AF2               BPL.S   WAIT_LOOP
2034
2035    0000099A 257C 0000          MOVE.L  #TMO_ERR,S_ERROR(A2) save the timeout error (Z:=0)
                    0011 0034
2036    000009A2 4E75               RTS
2037
2038                0000 09A4 WAIT_LOOP2    EQU *                      wait loop for infinite timeout
2039    000009A4 4AAA 0034          TST.L   S_ERROR(A2)             check for errors saved by ISRs
2040    000009A8 6604               BNE.S   EXIT_WAIT                  exit (Z=0)
2041    000009AA 4E94               JSR     (A4)
2042    000009AC 66F6               BNE.S   WAIT_LOOP2
2043
2044                0000 09AE EXIT_WAIT     EQU *
2045    000009AE 4E75               RTS
```

```
2047                        ********************************************************************************
2048                        *
2049                        *       CHECK_DSR_CTS, CHECK_QUEUE
2050                        *
2051                        *       FUNCTIONs to be used with WAIT, they all return Z=1 when the
2052                        *          condition is met.
2053                        *
2054                        *       USES:  the function is allowed to use only d6 and d7
2055                        *
2056                        ********************************************************************************
2057
2058                        *
2059                        *   condition:  queue is empty
2060                        *
2061             0000 09B0 CHECK_QUEUE     EQU *
2062    000009B0 3E2A 004A          MOVE.W  Q_OUT(A2),D7            (12)
2063    000009B4 BE6A 0048          CMP.W   Q_IN(A2),D7             (12)  Z=1 if empty
2064    000009B8 0A3C 0004          EORI    #$04,CCR                (20)  invert the Z bit (Z=1 if full)
2065    000009BC 4E75               RTS                             (----> 44 )
2066
2067                        *
2068                        *   condition:  DSR=1 and CTS=1  (ok to send with modem handshake)
2069                        *
2070             0000 09BE CHECK_DSR_CTS   EQU *
2071    000009BE 1E29 001D          MOVE.B  MODEM_STAT(A1),D7       (12)
2072    000009C2 8F2A 003E          OR.B    D7,S_MODEM(A2)          (16)  save modem status for user
2073    000009C6 4607               NOT.B   D7                      ( 4)
2074    000009C8 CE7C 0030          AND     #$30,D7                 ( 8)  if both DSR and CTS were true, Z=0
2075    000009CC 4E71               NOP                             ( 4)
2076    000009CE 4E75               RTS                             (----> 44 )
2077
```



```
2080                        ********************************************************************************
2081                        *
2082                        *       Buffer routines
2083                        *
2084                        ********************************************************************************
2085
2086                        *-------------------------------------------------------------------------------
2087                        *       INIT_QUEUE
2088                        *          initializes the queue descriptor.
2089                        *-------------------------------------------------------------------------------
2090
2091             0000 09D0 INIT_QUEUE      EQU *
2092    000009D0 357C 0088          MOVE.W  #BUFFER_SIZE,Q_SIZE(A2)         * initialize  queue_size
                 0046
2093    000009D6 426A 0048          CLR.W   Q_IN(A2)                        *  queue_in := 0
2094    000009DA 426A 004A          CLR.W   Q_OUT(A2)                       *  queue_out := 0
2095    000009DE 4E75               RTS
2096
2097                        *-------------------------------------------------------------------------------
2098                        *       QUEUE_EMPTY
2099                        *          tells if queue is empty.
2100                        *          ON EXIT:  Z=empty (IF EMPTY THEN Z:=1 ELSE Z:=0)
2101                        *          USES   :  D7
2102                        *-------------------------------------------------------------------------------
2103
2104             0000 09E0 QUEUE_EMPTY     EQU *
2105    000009E0 3E2A 004A          MOVE.W  Q_OUT(A2),D7
2106    000009E4 BE6A 0048          CMP.W   Q_IN(A2),D7             *  RETURN( queue_in = queue_out )
2107    000009E8 4E75               RTS
2108
2109                        *-------------------------------------------------------------------------------
2110                        *       QUEUE_FULL
2111                        *          tells if queue is full.
2112                        *          ON EXIT:  Z=full (IF FULL THEN Z:=1 ELSE Z:=0)
2113                        *          USES   :  D7
2114                        *-------------------------------------------------------------------------------
2115
2116             0000 09EA QUEUE_FULL      EQU *
2117    000009EA 3E2A 0048          MOVE.W  Q_IN(A2),D7             *
2118    000009EE 5247               ADDQ.W  #1,D7                   *
2119    000009F0 BE6A 004A          CMP.W   Q_OUT(A2),D7            *  queue_out = queue_in+1 ?
2120    000009F4 6602               BNE.S   CHECK_OR                *
2121    000009F6 4E75               RTS                             *        ( YES so return with Z=1)
2122             0000 09F8 CHECK_OR        EQU *
2123    000009F8 BE6A 0046          CMP.W   Q_SIZE(A2),D7           *  queue_in+1 = queue_size ?
2124    000009FC 6702               BEQ.S   CHECK_AND               *
2125    000009FE 4E75               RTS                             *        ( NO  so return with Z=0)
2126             0000 0A00 CHECK_AND       EQU *
2127    00000A00 3E2A 004A          MOVE.W  Q_OUT(A2),D7            *  queue_out = 0  ?
2128    00000A04 4E75               RTS                             *        ( ANSWER is result of function )
```

```
2130                        *-------------------------------------------------------------------------------
2131                        *       INQUEUE
2132                        *          puts a character in the queue
2133                        *          ON ENTRY:  d2.B - character to be put in the queue
2134                        *                     buffer NOT full !!
2135                        *          USES    :  a4.L - queue_addr
2136                        *                     d7.W - queue_in
2137                        *-------------------------------------------------------------------------------
2138
2139             0000 0A06 INQUEUE          EQU *
2140    00000A06 3E2A 0048          MOVE.W  Q_IN(A2),D7              *
2141    00000A0A 1582 704C          MOVE.B  D2,Q_BUFFER(A2,D7.W)     *  (queue_addr+queue_in)^ := char
2142
2143    00000A0E 5247               ADDQ.W  #1,D7                    *  queue_in := queue_in+1
2144
2145    00000A10 BE6A 0046          CMP.W   Q_SIZE(A2),D7            *
2146    00000A14 6C06               BGE.S   RESET_IN                 *  IF queue_in >= queue_size
2147    00000A16 3547 0048          MOVE.W  D7,Q_IN(A2)              *
2148    00000A1A 4E75               RTS                              *
2149             0000 0A1C RESET_IN         EQU *                    *
2150    00000A1C 426A 0048          CLR.W   Q_IN(A2)                 *    THEN queue_in := 0
2151    00000A20 4E75               RTS                              *
2152
2153                        *-------------------------------------------------------------------------------
2154                        *       OUTQUEUE
2155                        *          take the next character out of the queue
2156                        *          ON ENTRY:  buffer NOT full
2157                        *          ON EXIT :  d2.B - character from the queue
2158                        *          USES    :  a4.L - queue_addr
2159                        *                     d7.W - queue_in
2160                        *-------------------------------------------------------------------------------
2161
2162             0000 0A22 OUTQUEUE          EQU *
2163    00000A22 3E2A 004A          MOVE.W  Q_OUT(A2),D7             *
2164    00000A26 1432 704C          MOVE.B  Q_BUFFER(A2,D7.W),D2     *  char := (queue_addr+queue_out)^
2165
2166    00000A2A 5247               ADDQ.W  #1,D7                    *  queue_out := queue_out+1
2167
2168    00000A2C BE6A 0046          CMP.W   Q_SIZE(A2),D7            **
2169    00000A30 6C06               BGE.S   RESET_OUT                *  IF queue_out >= queue_size
2170    00000A32 3547 004A          MOVE.W  D7,Q_OUT(A2)             *
2171    00000A36 4E75               RTS                              *
2172             0000 0A38 RESET_OUT        EQU *                    *
2173    00000A38 426A 004A          CLR.W   Q_OUT(A2)                *    THEN queue_out := 0
2174    00000A3C 4E75               RTS                              **
```



```
2176                        *-------------------------------------------------------------------------------
2177                        *       QUEUE_SPACE
2178                        *          returns amount of space remaining in the queue
2179                        *          ON EXIT:  d3.W - contains the space remaining in the queue.
2180                        *-------------------------------------------------------------------------------
2181
2182             0000 0A3E QUEUE_SPACE      EQU *
2183    00000A3E 362A 004A          MOVE.W  Q_OUT(A2),D3             *
2184    00000A42 966A 0048          SUB.W   Q_IN(A2),D3              *  IF  queue_in >= queue_out
2185    00000A46 6E08               BGT.S   OUT_GREATER              *
2186
2187    00000A48 5343               SUBQ.W  #1,D3                    *  THEN queue_space :=
2188    00000A4A D66A 0046          ADD.W   Q_SIZE(A2),D3            *     queue_size + queue_out-queue_in - 1
2189    00000A4E 4E75               RTS
2190             0000 0A50 OUT_GREATER      EQU *
2191    00000A50 5343               SUBQ.W  #1,D3                    *  ELSE queue_space :=
2192    00000A52 4E75               RTS                              *     queue_out-queue_in - 1
2193                                END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ABORT_IO	ABS	109	00000002
IODECLARATIONS	ABS	246	00000007
LOGEOT	ABS	109	00000005
SYSGLOBALS	ABS	247	0000000B

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ABORT_MODEM	REL	1249		000004C2
ACK	ABS	499		00000006
ACK_CHAR	ABS	477		00000044
ACK_SIZE	ABS	502		00000050
ASM_INIT	REL	529		0000000C
AVAIL_OFF	ABS	186		00000034
BADTMO	ABS	264		0000000B
BAD_RDS	ABS	272		00000013
BAD_SCT	ABS	273		00000014
BAUD	REL	596		000000A0
BAUD_SW	ABS	437		00000005
BUFFER_SIZE	ABS	495		00000088
BUFI_OFF	ABS	177		00000024
BUFO_OFF	ABS	178		00000028
BUF_BUSY	ABS	262		00000009
BUSY_ERR	REL	1680		000007C4
CALC_DIV	REL	1090		000003B0
CCR	STREG	0		00000005
CHECK_AND	REL	2126		00000A00
CHECK_BREAK	REL	1337		00000590
CHECK_DSR_CTS	REL	2070		000009BE
CHECK_ERROR	REL	1795		00000874
CHECK_FHS	REL	1573		0000072C
CHECK_OR	REL	2122		000009F8
CHECK_QUEUE	REL	2061		000009B0
CHECK_XFER_IN	REL	1670		000007B4
CHECK_XFER_OUT	REL	1675		000007BC
CHECK_XIN	REL	1443		00000660
CHECK_XOFF	REL	1379		000005E6
CLEAR_OUT	REL	1644		00000796
CLEAR_XFER	REL	1637		00000780
CLR_XOUT	REL	1309		00000558
CONNECT	REL	1732		000007F6
CONNECTED	ABS	467		0000003A
CONT_0	REL	1049		00000374

CONT_1	REL	1054		0000037C
CONT_12	REL	1145		0000042A
CONT_13	REL	1154		0000043E
CONT_14	REL	1162		0000044E
CONT_15	REL	1169		00000458
CONT_16	REL	1173		0000045E
CONT_17	REL	1177		00000464
CONT_18	REL	1181		0000046A
CONT_19	REL	1185		00000470
CONT_3	REL	1075		000003A2
CONT_4	REL	1109		000003DE
CONT_5	REL	1130		00000412
CONT_6	REL	1134		00000418
CONT_7	REL	1140		00000422
CONT_ERROR	REL	1045		0000036E
CONT_TABLE	REL	1023		00000346
CONV_CHAR	ABS	478		00000045
CRD_DWN	ABS	274		00000015
C_ADR	ABS	176		00000020
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DATA	ABS	444		00000011
DC1	ABS	496		00000011
DC3	ABS	497		00000013
DISCONNECT	REL	1750		0000081A
DIV0	ABS	446		00000011
DIV1	ABS	447		00000013
DMA_ERR	REL	1506		000006B2
DONT_SET	REL	931		000002BE
DO_BIT_4	REL	809		000001C4
DO_BIT_5	REL	814		000001CE
DUMP_BUFFER	REL	1602		00000744
DUMP_LOOP	REL	1606		0000074A
EIRB_OFF	ABS	179		0000002C
END_ERR	REL	1505		000006B2
END_ISR	REL	1450		0000066E
END_RESTORE	REL	1936		0000092A
END_STS_2	REL	819		000001D8
END_XIN	REL	1616		00000762
END_XOUT	REL	1291		00000522
ENQ	ABS	498		00000005
ENQ_CHAR	ABS	476		00000043
ENQ_H	REL	1898		000008D4
ENQ_HAND	REL	1407		0000061A
EOD_SEEN	ABS	275		00000016
ERROR_EXIST	REL	1799		0000087C
ERROR_INTR	REL	1237		000004A4
ESC_CODE	ABS	283	SYSGLOBALS +	FFFFFFFE
EXIT_14	REL	1167		00000456
EXIT_C1	REL	1073		000003A0
EXIT_C4	REL	1128		00000410

EXIT_CONNECT	REL	1741		00000818
EXIT_DUMP	REL	1625		00000778
EXIT_TFR	REL	1581		00000742
EXIT_WAIT	REL	2044		000009AE
GET_CHAR	REL	1870		000008B4
HAND_TABLE	REL	1357		000005B2
H_ISR_PM	ABS	175		0000001C
H_ISR_PR	ABS	173		00000014
H_ISR_SL	ABS	174		00000018
H_TABLE	REL	1889		000008CC
ID_REG	ABS	435		00000001
INIT_QUEUE	REL	2091		000009D0
INPUT_END	REL	1433		00000652
INPUT_INTR	REL	1319		0000056C
INPUT_XFER	REL	1536		000006E0
INQUEUE	REL	2139		00000A06
INTR_EN	ABS	445		00000013
INTR_EXIST	REL	1219		0000048C
INTR_ID	ABS	448		00000015
INTR_SW	ABS	436		00000003
INTR_TABLE	REL	1226		0000049C
INTR_XFER	REL	1526		000006D4
IN_ISR	ABS	465		00000038
IOESCAPE	REL	1695		000007CA
IOE_ERROR	ABS	278		FFFFFFE6
IOE_RSLT	ABS	280	IODECLARATIONS +	FFFFFFBE
IOE_SC	ABS	281	IODECLARATIONS +	FFFFFFBA
IO_MISC	ABS	276		00000017
IO_SC	ABS	180		0000002D
IS21	REL	856		0000022A
IS43	REL	852		0000021E
IS77	REL	848		00000212
IS85	REL	844		00000206
ISR_ENTRY	ABS	171		00000000
LINE_CONT	ABS	449		00000017
LINE_STAT	ABS	451		0000001B
LINE_SW	ABS	438		00000007
LOOP_LAST	REL	1297		00000534
LOOP_THRE	REL	1958		00000936
MA	ABS	185		00000033
MA_W	ABS	184		00000032
MODEM_CONT	ABS	450		00000019
MODEM_INTR	REL	1243		000004B6
MODEM_ON	ABS	468		0000003B
MODEM_STAT	ABS	452		0000001D
MOVE_OUT	REL	1279		00000504
NOT_HPIB	ABS	255		00000002
NOT_ISR	REL	1980		00000960
NOT_LSTN	ABS	269		00000010
NOT_TALK	ABS	268		0000000F
NO_ACTL	ABS	256		00000003
NO_CARD	ABS	254		00000001
NO_CONVERT	REL	1348		000005A4
NO_DATA	ABS	259		00000006
NO_DMA	ABS	268		0000000D
NO_DRV	ABS	265		0000000C
NO_DVC	ABS	257		00000004

NO_HAND	REL	1427		00000648
NO_SCTL	ABS	271		00000012
NO_SPACE	ABS	258		00000005
NO_WORD	ABS	267		0000000E
OK_XMIT	REL	1830		0000089A
OUTPUT_INTR	REL	1255		000004D4
OUTPUT_XFER	REL	1557		00000706
OUTQUEUE	REL	2162		00000A22
OUT_GREATER	REL	2190		00000A50
OVERRUN	REL	1437		00000658
OVERRUN_ERROR	ABS	501		0000013A
PUTINQ	REL	1428		00000648
QUEUE_EMPTY	REL	2104		000009E0
QUEUE_FULL	REL	2116		000009EA
QUEUE_SPACE	REL	2182		00000A3E
Q_BUFFER	ABS	483		0000004C
Q_DESCRIPT	ABS	479		00000046
Q_IN	ABS	481		00000048
Q_OUT	ABS	482		0000004A
Q_SIZE	ABS	480		00000046
RCVR_BLK	ABS	284	SYSGLOBALS +	FFFFFFF6
RDIVU	REL	1710		000007E0
READ_END	REL	1939		0000092E
READ_UART	REL	898		0000027C
RECEIVING	ABS	469		0000003C
REGULAR	REL	860		00000236
REG_MAX	ABS	505		00000013
RESET_IN	REL	2149		00000A1C
RESET_OUT	REL	2172		00000A38
RESET_REG	ABS	434		00000001
RESTORE	REL	1925		00000904
ROUND	REL	1720		000007F0
RS_RS	REL	424		00000000
RS_RS_INIT	REL	514		00000002
RS_RS_ISR	REL	1196		00000476
RS_RS_RDB	REL	620		000000C0
RS_RS_RDS	REL	734		0000014A
RS_RS_RDW	REL	677		000000FE
RS_RS_TFR	REL	1461		00000674
RS_RS_WTB	REL	649		000000E0
RS_RS_WTC	REL	997		00000324
RS_RS_WTW	REL	704		00000124
SC_BUSY	ABS	261		00000008
SEND	REL	1953		00000930
SEND_ACK	REL	1416		00000634
SEND_HAND	REL	1920		000008F2
SER_FHS	REL	1520		000006C6
SET_BIT_0	REL	806		000001C0
SET_OUT	REL	1660		000007AE
SET_XFER	REL	1654		0000079C
SOFT_RESET	REL	1765		00000832
SP	AREG	0		00000007
SR	STREG	0		00000006
STS_0	REL	788		0000019E
STS_1	REL	792		000001A4
STS_10	REL	916		00000298
STS_11	REL	935		000002C2

STS_12	REL	949	000002E2
STS_13	REL	953	000002E8
STS_14	REL	957	000002EE
STS_15	REL	968	00000306
STS_16	REL	972	0000030C
STS_17	REL	976	00000312
STS_18	REL	980	00000318
STS_19	REL	984	0000031E
STS_2	REL	796	000001AA
STS_3	REL	823	000001DC
STS_4	REL	869	00000244
STS_5	REL	880	00000258
STS_6	REL	884	0000025E
STS_7	REL	902	00000282
STS_8	REL	908	0000028C
STS_9	REL	912	00000292
STS_ERROR	REL	784	00000198
STS_TABLE	REL	762	00000170
S_ERROR	ABS	464	00000034
S_HANDSH	ABS	473	00000040
S_LINE	ABS	472	0000003F
S_MODEM	ABS	471	0000003E
TACT_OFF	ABS	201	00000007
TBSZ_OFF	ABS	224	00000018
TBUF_OFF	ABS	223	00000014
TCHR_OFF	ABS	220	0000000E
TCNTERR	ABS	263	0000000A
TCNT_OFF	ABS	222	00000010
TDIR_OFF	ABS	218	0000000D
TEMP_OFF	ABS	225	0000001C
TEMP_SIZE	ABS	494	000000A0
TEND_OFF	ABS	216	0000000B
TERM_HAND	REL	1390	000005FA
TFIL_OFF	ABS	226	00000020
TFR_ERR	ABS	260	00000007
TIMEOUT	ABS	181	0000002E
TMO_ERR	ABS	270	00000011
TTMP_OFF	ABS	199	00000000
TT_BURST	ABS	237	00000003
TT_DMA	ABS	236	00000002
TT_FHS	ABS	238	00000004
TT_INT	ABS	235	00000001
TUSR_OFF	ABS	202	00000009
T_BW_OFF	ABS	214	0000000A
T_DMAPRI	ABS	231	00000030
T_PM_OFF	ABS	230	0000002C
T_PR_OFF	ABS	227	00000024
T_SC_OFF	ABS	200	00000005
T_SL_OFF	ABS	229	00000028
UNDERSCORE	ABS	500	0000005F
USER_ISR	ABS	172	00000014
USP	STREG	0	00000007
VAL_ERR	REL	1150	00000438
WAIT	REL	2011	00000974
WAIT_BREAK	REL	1062	0000038C
WAIT_BREAK2	REL	1070	0000039C
WAIT_FHS	REL	1578	0000073A

WAIT_GET	REL	1845	000008A2
WAIT_LOOP	REL	2026	0000098C
WAIT_LOOP2	REL	2038	000009A4
WAIT_SEND	REL	1819	00000894
WORD_ERR	REL	1512	000006BC
XFER_ERR	REL	1507	000006B2
XFER_OUT	REL	1267	000004E4
XFER_TABLE	REL	1489	0000069E
XIN_ACT	ABS	466	00000039
XMITTING	ABS	470	0000003D
XMIT_CHAR	REL	1988	0000096C
XOFF_CHAR	ABS	475	00000042
XOFF_SIZE	ABS	503	00000028
XON_CHAR	ABS	474	00000041
XON_H	REL	1911	000008E4
XON_HAND	REL	1363	000005BA
XON_SIZE	ABS	504	00000060

RSTRINT

Purpose

RSTRINT implements the conversion of a string of digits into an integer.

Usage

The Compiler emits a call to this routine for STRREAD of an integer.

```
 PAGE   1 [2.0] 11/4/82 13:54:29  ASSEMBLY OF RSTRINT.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                                   def     fs_freadstrint
    2                                   refa    sysglobals
    3
    4               FFFF FFEA ioresult          equ sysglobals-22
    5               0000 000E ibadformat        equ 14          improper syntax for an integer
    6
    7               0000 0000 error             equ D0          ioresult
    8               0000 0001 remainder         equ D1          characters left in string
    9               0000 0002 index             equ D2          subscript into string
   10               0000 0003 char              equ D3          character in question
   11               0000 0004 int               equ D4          integer being built
   12               0000 0005 sign              equ D5          minus flag
   13               0000 0006 oldindex          equ D6          original value of index
   14               0000 0007 temp              equ D7
   15
   16               0000 0000 return            equ A0          subroutine return address
   17               0000 0001 I                 equ A1          address of integer to be returned
   18               0000 0002 aindex            equ A2          address of index into string
   19               0000 0003 string            equ A3          address of source string
   20
   21               0000 0000 fs_freadstrint    equ *
   22     00000000 205F               movea.l (sp)+,return
   23     00000002 225F               movea.l (sp)+,I
   24     00000004 245F               movea.l (sp)+,aindex
   25     00000006 265F               movea.l (sp)+,string
   26
   27     00000008 7600               moveq   #0,char                 sign extend digits
   28     0000000A 7200               moveq   #0,remainder
   29     0000000C 1213               move.b  (string),remainder      get current length of string
   30     0000000E 2412               move.l  (aindex),index          get subscript into it
   31     00000010 2C02               move.l  index,oldindex          save in case of error
   32     00000012 6F06               ble.s   L0                      error if < 1
   33     00000014 D6C2               adda    index,string            advance to first interesting character
   34     00000016 9282               sub.l   index,remainder         how many characters have we left?
   35     00000018 6C0A               bge.s   L1
   36     0000001A 700E       L0      moveq   #ibadformat,error      index is past end of string!
   37     0000001C 6058               bra.s   endit
   38
   39     0000001E 5242       L2      addq    #1,index                bump user index
   40     00000020 5341               subq    #1,remainder            any more characters?
   41     00000022 6DF6               blt.s   L0
   42     00000024 161B       L1      move.b  (string)+,char
   43     00000026 B63C 0020          cmp.b   #' ',char               skip spaces
   44     0000002A 67F2               beq.s   L2
   45
   46     0000002C 50C0               st      error                   assume error until see digit
   47     0000002E 7800               moveq   #0,int                  initialize value
   48
   49     00000030 B63C 002D          cmp.b   #'-',char               is it a minus?
   50     00000034 57C5               seq     sign
   51     00000036 6706               beq.s   L3
   52     00000038 B63C 002B          cmp.b   #'+',char               is it a plus?
   53     0000003C 6608               bne.s   L4
   54
   55     0000003E 5242       L3      addq    #1,index                bump user index
   56     00000040 5341               subq    #1,remainder            any more characters?
   57     00000042 6D24               blt.s   L5
   58     00000044 161B               move.b  (string)+,char
```

```
 PAGE   2 [2.0] 11/4/82 13:54:29  ASSEMBLY OF RSTRINT.TEXT

   59     00000046 963C 0030 L4       sub.b   #'0',char               is it a digit
   60     0000004A 6D1C               blt.s   L5
   61     0000004C B63C 0009          cmp.b   #9,char
   62     00000050 6E16               bgt.s   L5
   63     00000052 51C0               sf      error                   no error, at least one digit
   64
   65     00000054 D884               add.l   int,int                 multiply integer by 10
   66     00000056 69C2               bvs.s   L0
   67     00000058 2E04               move.l  int,temp
   68     0000005A E584               asl.l   #2,int
   69     0000005C 69BC               bvs.s   L0
   70     0000005E D887               add.l   temp,int
   71     00000060 69B8               bvs.s   L0
   72
   73     00000062 9883               sub.l   char,int                add value of digit
   74     00000064 68D8               bvc.s   L3                      go back for more
   75     00000066 60B2               bra.s   L0
   76
   77     00000068 4A00       L5      tst.b   error                   were there any digits?
   78     0000006A 66AE               bne.s   L0
   79     0000006C 4A05               tst.b   sign                    was it positive
   80     0000006E 6604               bne.s   L8
   81     00000070 4484               neg.l   int                     make it positive
   82     00000072 69A6               bvs.s   L0
   83     00000074 7000       L8      moveq   #0,error                all done, no errors
   84
   85
   86     00000076 2B40 FFEA endit    move.l  error,ioresult(a5)
   87     0000007A 6704               beq.s   ok
   88     0000007C 2406               move.l  oldindex,index
   89     0000007E 7800               moveq   #0,int
   90     00000080 2482       ok      move.l  index,(aindex)
   91     00000082 2284               move.l  int,(I)
   92     00000084 4ED0               jmp     (return)
   93
   94                                 nosyms
   95                                 end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

SCAN

Purpose

This routine scans a contiguous area of memory, comparing each byte against a test character.

Usage

The Compiler emits calls to this routine.

```
 PAGE   1 [2.0] 11/4/82 13:54:50  ASSEMBLY OF SCAN.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                           ********************************************************************************
    2                           *
    3                           *       SCAN function -- equivalent to UCSD SCAN
    4                           *         --written for M68000 by John Schmidt, 10/80
    5                           *
    6                           *       Parameters are:
    7                           *        (SP): return address
    8                           *       4(SP): count -- may be < 0, long word
    9                           *       8(SP): character to match
   10                           *      10(SP): address to start scan
   11                           *      14(SP): =/<> flag coded as 0/1
   12                           *
   13                           *  Note: no range checking is performed within this function !
   14                           *
   15                           ********************************************************************************
   16                           *
   17                                    DEF ASM_SCAN
   18    00000000 285F          ASM_SCAN MOVEA.L (SP)+,A4          GET RETURN ADDRESS
   19    00000002 201F                   MOVE.L  (SP)+,D0          GET COUNT
   20    00000004 121F                   MOVE.B  (SP)+,D1          GET CHARACTER TO MATCH
   21    00000006 205F                   MOVEA.L (SP)+,A0          GET START ADDRESS OF SCAN
   22    00000008 341F                   MOVE    (SP)+,D2          GET =/<> FLAG WORD
   23    0000000A 7600                   MOVEQ   #0,D3             INITIALIZE RESULT
   24    0000000C 4A80                   TST.L   D0                CHECK FOR ZERO COUNT
   25    0000000E 673A                   BEQ.S   EXIT              RETURN IF COUNT=0
   26    00000010 6D1C                   BLT.S   BACKSCAN          IF NEGATIVE COUNT SCAN BACKWARDS
   27    00000012 4A42                   TST.W   D2                CHECK =/<> FLAG TO SELECT LOOP
   28    00000014 660C                   BNE.S   SC@N2
   29    00000016 B218          SC@N1    CMP.B   (A0)+,D1          LOOK FOR MATCH
   30    00000018 6730                   BEQ.S   EXIT              IF FOUND THEN DONE
   31    0000001A 5283                   ADDQ.L  #1,D3             BUMP RESULT
   32    0000001C 5380                   SUBQ.L  #1,D0             KEEP COUNT
   33    0000001E 66F6                   BNE     SC@N1
   34    00000020 6028                   BRA.S   EXIT              IF DONE THEN GET OUT
   35    00000022 B218          SC@N2    CMP.B   (A0)+,D1          THIS LOOP IS JUST LIKE THE ONE ABOVE
   36    00000024 6624                   BNE.S   EXIT              EXCEPT FOR THIS TEST
   37    00000026 5283                   ADDQ.L  #1,D3
   38    00000028 5380                   SUBQ.L  #1,D0
   39    0000002A 66F6                   BNE     SC@N2
   40    0000002C 601C                   BRA.S   EXIT
   41    0000002E 5288          BACKSCAN ADDQ.L  #1,A0             KLUDGE ADDRESS SINCE WE'LL DECREMENT IN LOOP
   42    00000030 4A42                   TST.W   D2
   43    00000032 660C                   BNE.S   SC@N4             CHECK FOR <> BACKWARD SCAN
   44    00000034 B220          SC@N3    CMP.B   -(A0),D1          LOOK FOR MATCH
   45    00000036 6712                   BEQ.S   EXIT              DONE IF FOUND
   46    00000038 5383                   SUBQ.L  #1,D3             DECREMENT RESULT
   47    0000003A 5280                   ADDQ.L  #1,D0             KEEP COUNT
   48    0000003C 66F6                   BNE     SC@N3
   49    0000003E 600A                   BRA.S   EXIT
   50    00000040 B220          SC@N4    CMP.B   -(A0),D1
   51    00000042 6606                   BNE.S   EXIT
   52    00000044 5383                   SUBQ.L  #1,D3
   53    00000046 5280                   ADDQ.L  #1,D0
   54    00000048 66F6                   BNE     SC@N4
   55    0000004A 2F03          EXIT     MOVE.L  D3,-(SP)          PUT RESULT ON THE STACK
   56    0000004C 4ED4                   JMP (A4)                  GOTO RETURN ADDRESS
   57                                    END
PASS 1 ERRORS: 0
```

```
 PAGE   2 [2.0] 11/4/82 13:54:50  ASSEMBLY OF SCAN.TEXT

PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

*** NO EXTERNAL SYMBOLS ***

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ASM_SCAN	REL	18		00000000
BACKSCAN	REL	41		0000002E
CCR	STREG	0		00000005
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
EXIT	REL	55		0000004A
SC@N1	REL	29		00000016
SC@N2	REL	35		00000022
SC@N3	REL	44		00000034
SC@N4	REL	50		00000040
SP	AREG	0		00000007
SR	STREG	0		00000006
USP	STREG	0		00000007

SETSTUFF

Purpose

SETSTUFF contains assembly language to perform set assignment and adding an element to a set.

Usage

The Compiler emits calls to these routines.

```
PASS 1 COMPLETE. ERRORS: 0
    1                           *----------------------------------------
    2                                       DEF asm_SETASSIGN
    3                           *----------------------------------------
    4               0000 0000 asm_SETASSIGN    EQU *
    5                           * obtain sets from stack
    6     00000000 266F 0004                movea.l    4(sp),a3          address of source
    7     00000004 286F 0008                movea.l    8(sp),a4          address of dest
    8                           * place size in d7
    9     00000008 3E1B                     move.w     (a3)+,d7          size of source
   10     0000000A 3887                     move.w     d7,(a4)           store size in dest
   11     0000000C 6766                     beq.s      done2             check for zero length set
   12     0000000E 302F 000C                move.w     12(sp),d0         dest min
   13     00000012 672C                     beq.s      testhigh
   14     00000014 7400                     moveq      #0,d2             zero count
   15     00000016 3607                     move.w     d7,d3
   16     00000018 244B                     movea.l    a3,a2
   17     0000001A 4A1A       loop1         tst.b      (a2)+             find nonzero byte
   18     0000001C 660A                     bne.s      NZbyte
   19     0000001E 5242                     addq.w     #1,d2
   20     00000020 5343                     subq.w     #1,d3
   21     00000022 6EF6                     bgt.s      loop1
   22
   23     00000024 4254                     clr.w      (a4)
   24     00000026 604C                     bra.s      done2
   25
   26     00000028 E74A       NZbyte        lsl.w      #3,d2             byte count * 8
   27     0000002A B042                     cmp.w      d2,d0
   28     0000002C 6F12                     ble.s      testhigh
   29
   30     0000002E 162A FFFF                move.b     -1(a2),d3         get first nonzero byte
   31     00000032 E30B       loop2         lsl.b      #1,d3
   32     00000034 6504                     bcs.s      NZbit             nonzero bit found
   33     00000036 5242                     addq.w     #1,d2
   34     00000038 60F8                     bra.s      loop2
   35
   36     0000003A B042       NZbit         cmp.w      d2,d0
   37     0000003C 6F02                     ble.s      testhigh
   38     0000003E 4E47                     trap       #7                error
   39
   40     00000040 322F 000E testhigh       move.w     14(sp),d1         dest max
   41     00000044 3633 70FE lastword       move.w     -2(a3,d7.w),d3    get last word of set
   42     00000048 6608                     bne.s      NZword
   43     0000004A 5547                     subq.w     #2,d7
   44     0000004C 6EF6                     bgt.s      lastword
   45     0000004E 4254                     clr.w      (a4)              set was empty
   46     00000050 6022                     bra.s      done2
   47
   48     00000052 3407       NZword        move.w     d7,d2
   49     00000054 E74A                     lsl.w      #3,d2             byte count * 8
   50     00000056 5342       loop3         subq.w     #1,d2
   51     00000058 E24B                     lsr.w      #1,d3
   52     0000005A 64FA                     bcc.s      loop3
   53
   54     0000005C B242                     cmp.w      d2,d1             last nonzero bit found
   55     0000005E 6C02                     bge.s      ok
   56     00000060 4E47                     trap       #7                error
   57                          * perform assignment
   58     00000062 38C7       ok            move.w     d7,(a4)+           store size in dest
```



```
   59     00000064 E447                     asr.w      #2,d7              determine size in long words
   60     00000066 6406                     bcc.s      evenn                even number of long words
   61     00000068 38DB                     move.w     (a3)+,(a4)+        move "odd" word
   62     0000006A 4A47                     tst.w      d7                 min size single word?
   63     0000006C 6706                     beq.s        done2
   64     0000006E 28DB       EVENN         move.l     (a3)+,(a4)+        move long words
   65     00000070 5347                     subq.w     #1,d7
   66     00000072 6EFA                     bgt.s        evenn
   67     00000074 205F       DONE2         movea.l    (sp)+,a0            eliminate extra bytes in stack
   68     00000076 DFFC 0000                adda.l     #12,sp
                   000C
   69     0000007C 4ED0                     jmp        (a0)
```

```
                                *----------------------------------------
                                        DEF asm_aDELEMENT
                                *----------------------------------------
                  0000 007E asm_aDELEMENT EQU         *
0000007E 205F                           movea.l         (sp)+,a0        return address
00000080 201F                           move.l          (sp)+,d0        element number to add to set
00000082 0C80 0000                      cmpi.l          #255,d0
                  00FF
00000088 6302                           bls.s           strt
0000008A 4E47                               trap    #7
0000008C 225F                 strt      movea.l         (sp)+,a1        source address
0000008E 2457                           movea.l         (sp),a2         destination address
00000090 3E19                           move.w          (a1)+,d7        get set size of source
00000092 34C7                           move.w          d7,(a2)+        store size value
00000094 B3CA                           cmpa.l          a2,a1           see if source and destination are equal
00000096 670C                           beq.s           insert
                                * copy source set to the destination set
00000098 264A                 setcopy   movea.l         a2,a3           save destination address
0000009A 3C07                           move.w          d7,d6           save size for destination
0000009C 6F06                           ble.s           insert          check for size of zero
0000009E 36D9                 rept      move.w          (a1)+,(a3)+     sets are always an even number of bytes
000000A0 5546                           subq.w          #2,d6
000000A2 6EFA                           bgt.s           rept
                                * insert an element in a set, adjusting the size of the destination if needed
000000A4 48C0                 insert    ext.l           d0
000000A6 81FC 0010                      divs            #16,d0           byte offset in low word
000000AA 2A00                           move.l          d0,d5
000000AC 4845                           swap            d5              bit offset from left of byte
000000AE 9A7C 000F                      sub.w           #15,d5
000000B2 4445                           neg.w           d5              bit offset from right
000000B4 E340                           asl             #1,d0           make d0 a byte offset
000000B6 3200                           move.w          d0,d1           compute final size into d1
000000B8 5441                           addq.w          #2,d1           put zeros in the two bytes containing
000000BA 3401                           move.w          d1,d2           the new bit if it is beyond current size
000000BC 926A FFFE                      sub.w           -2(a2),d1
000000C0 6F0E                           ble.s           exxiit
000000C2 3542 FFFE                      move.w          d2,-2(a2)       store appropriate size for set
000000C6 47F2 2000                      lea             0(a2,d2),a3
000000CA 4263                 zerout    clr.w           -(a3)
000000CC 5541                           subq.w          #2,d1
000000CE 6EFA                           bgt.s           zerout
000000D0 0885 0003 exxiit               bclr            #3,d5           { received upgrade 9/9 }
000000D4 6706                           beq.s           skiipp
000000D6 0BF2 0000                      bset            d5,0(a2,d0)
000000DA 4ED0                           jmp             (a0)
000000DC 0BF2 0001 skiipp               bset            d5,1(a2,d0)
000000E0 4ED0                           jmp             (a0)
                                *----------------------------------------
                                        end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

*** NO EXTERNAL SYMBOLS ***

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ASM_ADELEMENT	REL	74		0000007E
ASM_SETASSIGN	REL	4		00000000
CCR	STREG	0		00000005
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DONE2	REL	67		00000074
EVENN	REL	64		0000006E
EXXIIT	REL	111		000000D0
INSERT	REL	94		000000A4
LASTWORD	REL	41		00000044
LOOP1	REL	17		0000001A
LOOP2	REL	31		00000032
LOOP3	REL	50		00000056
NZBIT	REL	36		0000003A
NZBYTE	REL	26		00000028
NZWORD	REL	48		00000052
OK	REL	58		00000062
REPT	REL	90		0000009E
SETCOPY	REL	87		00000098
SKIIPP	REL	115		000000DC
SP	AREG	0		00000007
SR	STREG	0		00000006
STRT	REL	80		0000008C
TESTHIGH	REL	40		00000040
USP	STREG	0		00000007
ZEROUT	REL	108		000000CA

STRG1

Purpose

STRG1 contains assembly language routines to support these Pascal routines:

- STRLTRIM
- STRRTRIM
- STRRPT
- STRMOVE

Usage

The Compiler emits calls to these routines.

```
 PAGE   1 [2.0] 11/4/82 13:55:07  ASSEMBLY OF STRG1.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                0000 0000 RET      EQU     A0       return address
    2                0000 0001 SADDR    EQU     A1       source address
    3                0000 0002 DADDR    EQU     A2       destination address
    4                0000 0003 SRCEND   EQU     A3       source end
    5                0000 0004 ATEMP    EQU     A4       temporary A register
    6                          *
    7                0000 0000 SLEN     EQU     D0       source length
    8                0000 0001 SINDEX   EQU     D1       source index
    9                0000 0002 CURSLEN  EQU     D2       current source length
   10                0000 0003 DINDEX   EQU     D3       destination index
   11                0000 0004 DMAX     EQU     D4       destination max
   12                0000 0005 COUNT    EQU     D5
   13                0000 0005 INDEX    EQU     D5       string index
   14                0000 0006 TEMP1    EQU     D6       temporary D register
   15                0000 0007 TEMP2    EQU     D7       temporary D register
   16                          *
   17                                    refa    SYSGLOBALS
   18                FFFF FFFE ESCAPECODE EQU   SYSGLOBALS-2
   19                          *
   20                          *
   21                          *
   22                          *  procedure psubtopsub(dsublen: integer;
   23                          *                       var dsub: paoctype;
   24                          *                       dindex: integer;
   25                          *                       ssublen: integer;
   26                          *                       var ssub: paoctype;
   27                          *                       sindex,slen: integer);
   28                          *
   29                0000 0000 ASM_PSUBTOPSUB  EQU *
   30                                    def     ASM_PSUBTOPSUB
   31                          *
   32    00000000 205F               movea.l (sp)+,RET
   33    00000002 201F               move.l  (sp)+,SLEN
   34    00000004 6E08               bgt.s   LLLL1
   35    00000006 DFFC 0000                  adda.l     #24,sp
                  0018
   36    0000000C 4ED0                       jmp        (RET)
   37    0000000E 221F       LLLL1   move.l  (sp)+,SINDEX
   38    00000010 6F00 0088          ble     error
   39
   40    00000014 225F               movea.l (sp)+,SADDR
   41    00000016 241F               move.l  (sp)+,CURSLEN
   42    00000018 D4BC 0000          add.l   #1,CURSLEN
                  0001
   43    0000001E 2C01               move.l  SINDEX,TEMP1
   44    00000020 DC80               add.l   SLEN,TEMP1
   45    00000022 B486               cmp.l   TEMP1,CURSLEN
   46    00000024 6D74               blt.s   error
   47
   48    00000026 261F               move.l  (sp)+,DINDEX
   49    00000028 6F70               ble.s   error
   50
   51    0000002A 2C03               move.l  DINDEX,TEMP1
   52    0000002C DC80               add.l   SLEN,TEMP1
   53    0000002E 245F               movea.l (sp)+,DADDR
   54    00000030 281F               move.l  (sp)+,DMAX
   55    00000032 D8BC 0000          add.l   #1,DMAX
                  0001
```

```
 PAGE   2 [2.0] 11/4/82 13:55:07  ASSEMBLY OF STRG1.TEXT

   56    00000038 B886               cmp.l   TEMP1,DMAX
   57    0000003A 6D5E               blt.s   error
   58
   59    0000003C 43F1 10FF          lea     -1(SADDR,SINDEX.W),SADDR
   60    00000040 45F2 30FF          lea     -1(DADDR,DINDEX.W),DADDR
   61    00000044 6000 00A0          bra     TRANSFER
   62                          *
   63                          *
   64                          *
   65                          *  procedure ssubtopsub(dsublen: integer;
   66                          *                       var dsub: paoctype;
   67                          *                       dindex: integer;
   68                          *                       var ssub: string;
   69                          *                       sindex,slen: integer);
   70                          *
   71                0000 0048 ASM_SSUBTOPSUB  EQU *
   72                                    def     ASM_SSUBTOPSUB
   73                          *
   74    00000048 205F               movea.l (sp)+,RET
   75    0000004A 201F               move.l  (sp)+,SLEN
   76    0000004C 6E08               bgt.s   LLL1
   77    0000004E DFFC 0000                  adda.l     #22,sp
                  0016
   78    00000054 4ED0                       jmp        (RET)
   79    00000056 221F       LLL1    move.l  (sp)+,SINDEX
   80    00000058 6F40               ble.s   error
   81
   82    0000005A 225F               movea.l (sp)+,SADDR
   83    0000005C 7400               moveq   #0,CURSLEN
   84    0000005E 1411               move.b  (SADDR),CURSLEN
   85    00000060 5242               addq.w  #1,CURSLEN
   86    00000062 2C01               move.l  SINDEX,TEMP1
   87    00000064 DC80               add.l   SLEN,TEMP1
   88    00000066 B486               cmp.l   TEMP1,CURSLEN
   89    00000068 6D30               blt.s   error
   90
   91    0000006A 3C1F               move.w  (sp)+,TEMP1
   92    0000006C 261F               move.l  (sp)+,DINDEX
   93    0000006E 6F2A               ble.s   error
   94
   95    00000070 2C03               move.l  DINDEX,TEMP1
   96    00000072 DC80               add.l   SLEN,TEMP1
   97    00000074 245F               movea.l (sp)+,DADDR
   98    00000076 281F               move.l  (sp)+,DMAX
   99    00000078 5284               addq.l  #1,DMAX
  100    0000007A B886               cmp.l   TEMP1,DMAX
  101    0000007C 6D1C               blt.s   error
  102
  103    0000007E 43F1 1000          lea     0(SADDR,SINDEX.W),SADDR
  104    00000082 45F2 30FF          lea     -1(DADDR,DINDEX.W),DADDR
  105    00000086 605E               bra.s   TRANSFER
  106                         *
  107                         *
  108                         *
  109                         *  procedure ssubtossub(var dsub: string;
  110                         *                       dindex: integer;
  111                         *                       var ssub: string;
```

```
                        *                       sindex,slen: integer);
                        *
            0000 0088 ASM_SSUBTOSSUB  EQU *
                              def     ASM_SSUBTOSSUB
                        *
00000088 205F                 movea.l (sp)+,RET
0000008A 201F                 move.l  (sp)+,SLEN
0000008C 6E08                 bgt.s   L1
0000008E DFFC 0000                adda.l      #20,sp
         0014
00000094 4ED0                     jmp         (RET)
00000096 221F         L1      move.l  (sp)+,SINDEX
00000098 6E08                 bgt.s   L2
0000009A 3B7C FFF8 error          move.w      #-8,ESCAPECODE(a5)
         FFFE
000000A0 4E4A                     trap        #10
000000A2 225F         L2      movea.l (sp)+,SADDR
000000A4 7400                 moveq   #0,CURSLEN
000000A6 1411                 move.b  (SADDR),CURSLEN
000000A8 5242                 addq.w  #1,CURSLEN
000000AA 2C01                 move.l  SINDEX,TEMP1
000000AC DC80                 add.l   SLEN,TEMP1
000000AE B486                 cmp.l   TEMP1,CURSLEN
000000B0 6DE8                 blt.s   error

000000B2 3C1F                 move.w  (sp)+,TEMP1
000000B4 261F                 move.l  (sp)+,DINDEX
000000B6 6FE2                 ble.s   error

000000B8 2C03                 move.l  DINDEX,TEMP1
000000BA DC80                 add.l   SLEN,TEMP1
000000BC 245F                 movea.l (sp)+,DADDR
000000BE 7800                 moveq   #0,DMAX
000000C0 181F                 move.b  (sp)+,DMAX
000000C2 7E00                 moveq   #0,TEMP2
000000C4 3E04                 move.w  DMAX,TEMP2
000000C6 5287                 addq.l  #1,TEMP2
000000C8 BE86                 cmp.l   TEMP1,TEMP2
000000CA 6DCE                 blt.s   error

000000CC 7E00                 moveq   #0,TEMP2
000000CE 1E12                 move.b  (DADDR),TEMP2
000000D0 5287                 addq.l  #1,TEMP2
000000D2 BE83                 cmp.l   DINDEX,TEMP2
000000D4 6DC4                 blt.s   error

000000D6 BE46                 cmp.w   TEMP1,TEMP2
000000D8 6C04                 bge.s   L3
000000DA 5346                 subq.w  #1,TEMP1
000000DC 1486                 move.b  TEMP1,(DADDR)
                        *
                        *  End of error checking.
                        *
000000DE 43F1 1000    L3      lea     0(SADDR,SINDEX.W),SADDR
000000E2 45F2 3000            lea     0(DADDR,DINDEX.W),DADDR

            0000 00E6 TRANSFER equ *
```



```
000000E6 B5C9                 cmpa.l  SADDR,DADDR
000000E8 6E08                 bgt.s   topdown
                        *
                        *  Transfer from bottom to top
                        *
000000EA 14D9         L4      move.b  (SADDR)+,(DADDR)+
000000EC 5380                 subq.l  #1,SLEN
000000EE 6EFA                 bgt.s   L4
000000F0 4ED0                 jmp     (RET)
                        *
                        *  Transfer from top to bottom
                        *
            0000 00F2 topdown equ *
000000F2 43F1 0000            lea     0(SADDR,SLEN),SADDR
000000F6 45F2 0000            lea     0(DADDR,SLEN),DADDR
000000FA 1521         L5      move.b  -(SADDR),-(DADDR)
000000FC 5380                 subq.l  #1,SLEN
000000FE 6EFA                 bgt.s   L5
00000100 4ED0                 jmp     (RET)
                        *
                        *
                        *
                        *  procedure psubtossub(var dsub: string;
                        *                       dindex: integer;
                        *                       ssublen: integer;
                        *                       var ssub: paoctype;
                        *                       sindex,slen: integer);
                        *
            0000 0102 ASM_PSUBTOSSUB  EQU *
                              def     ASM_PSUBTOSSUB
                        *
00000102 205F                 movea.l (sp)+,RET
00000104 201F                 move.l  (sp)+,SLEN
00000106 6E08                 bgt.s   LL1
00000108 DFFC 0000                adda.l      #22,sp
         0016
0000010E 4ED0                     jmp         (RET)
00000110 221F         LL1     move.l  (sp)+,SINDEX
00000112 6F86                 ble.s   error

00000114 225F                 movea.l (sp)+,SADDR
00000116 241F                 move.l  (sp)+,CURSLEN
00000118 5282                 addq.l  #1,CURSLEN
0000011A 2C01                 move.l  SINDEX,TEMP1
0000011C DC80                 add.l   SLEN,TEMP1
0000011E B486                 cmp.l   TEMP1,CURSLEN
00000120 6D00 FF78            blt     error

00000124 261F                 move.l  (sp)+,DINDEX
00000126 6F00 FF72            ble     error

0000012A 2C03                 move.l  DINDEX,TEMP1
0000012C DC80                 add.l   SLEN,TEMP1
0000012E 245F                 movea.l (sp)+,DADDR
00000130 7800                 moveq   #0,DMAX
00000132 181F                 move.b  (sp)+,DMAX
00000134 7E00                 moveq   #0,TEMP2
```

```
00000136 3E04              move.w  DMAX,TEMP2
00000138 5287              addq.l  #1,TEMP2
0000013A BE86              cmp.l   TEMP1,TEMP2
0000013C 6D00 FF5C         blt     error

00000140 7E00              moveq   #0,TEMP2
00000142 1E12              move.b  (DADDR),TEMP2
00000144 5287              addq.l  #1,TEMP2
00000146 BE83              cmp.l   DINDEX,TEMP2
00000148 6D00 FF50         blt     error

0000014C BE46              cmp.w   TEMP1,TEMP2
0000014E 6C04              bge.s   LL2
00000150 5346              subq.w  #1,TEMP1
00000152 1486              move.b  TEMP1,(DADDR)

00000154 43F1 10FF LL2     lea     -1(SADDR,SINDEX.W),SADDR
00000158 45F2 3000         lea     0(DADDR,DINDEX.W),DADDR
0000015C 6088              bra.s   TRANSFER
                      *
                      *
                      *
                      *  function strrtrim(s: stringmax): stringmax;
                      *
                      *
          0000 015E ASM_STRRTRIM  EQU *
                              DEF   ASM_STRRTRIM
                      *
0000015E 205F              movea.l (sp)+,RET
00000160 225F              movea.l (sp)+,SADDR
00000162 245F              movea.l (sp)+,DADDR
00000164 7000              moveq   #0,SLEN
00000166 1019              move.b  (SADDR)+,SLEN
00000168 47F1 0000         lea     0(SADDR,SLEN.W),SRCEND
0000016C 5240              addq.w  #1,SLEN
0000016E 5340      rtrim1  subq.w  #1,SLEN
00000170 6F06              ble.s   null
00000172 0C23 0020         cmpi.b  #32,-(SRCEND)
00000176 67F6              beq.s   rtrim1

00000178 14C0      null    move.b  SLEN,(DADDR)+
0000017A 6706              beq.s   retrn

0000017C 14D9      rtrim2  move.b  (SADDR)+,(DADDR)+
0000017E 5340              subq.w  #1,SLEN
00000180 6EFA              bgt.s   rtrim2
00000182 4ED0      retrn   jmp     (RET)
                      *
                      *
                      *
                      *  function strltrim(s: stringmax): stringmax;
                      *
                      *
          0000 0184 ASM_STRLTRIM  EQU *
                              DEF   ASM_STRLTRIM
                      *
00000184 205F              movea.l (sp)+,RET
```



```
00000186 225F              movea.l (sp)+,SADDR
00000188 245F              movea.l (sp)+,DADDR
0000018A 7000              moveq   #0,SLEN
0000018C 1019              move.b  (SADDR)+,SLEN
0000018E 670C              beq.s   ltrim2
00000190 6004              bra.s   ltrim1

00000192 5340      ltrim0  subq.w  #1,SLEN
00000194 6706              beq.s   ltrim2

00000196 0C19 0020 ltrim1  cmpi.b  #32,(SADDR)+
0000019A 67F6              beq.s   ltrim0

0000019C 14C0      ltrim2  move.b  SLEN,(DADDR)+
0000019E 670A              beq.s   retrnn

000001A0 43E9 FFFF         lea     -1(SADDR),SADDR
000001A4 14D9      ltrim3  move.b  (SADDR)+,(DADDR)+
000001A6 5340              subq.w  #1,SLEN
000001A8 6EFA              bgt.s   ltrim3
000001AA 4ED0      retrnn  jmp     (RET)
                      *
                      *
                      *
                      *  function strrpt(s: stringmax; count: integer)
                      *                                          : stringmax;
                      *
                      *
          0000 01AC ASM_STRRPT    EQU *
                              DEF   ASM_STRRPT
                      *
000001AC 205F              movea.l (sp)+,RET
000001AE 2A1F              move.l  (sp)+,COUNT
000001B0 225F              movea.l (sp)+,SADDR
000001B2 245F              movea.l (sp)+,DADDR
000001B4 7000              moveq   #0,SLEN
000001B6 1019              move.b  (SADDR)+,SLEN
000001B8 2C05              move.l  COUNT,TEMP1
000001BA 0C86 0000         cmpi.l  #255,TEMP1
          00FF
000001C0 6200 FED8         bhi     error

000001C4 CDC0              muls    SLEN,TEMP1
000001C6 0C86 0000         cmpi.l  #255,TEMP1
          00FF
000001CC 6E00 FECC         bgt     error

000001D0 14C6              move.b  TEMP1,(DADDR)+
000001D2 670E              beq.s   retrnn

000001D4 2849      rpt0    movea.l SADDR,ATEMP
000001D6 2C00              move.l  SLEN,TEMP1
000001D8 14DC      rpt1    move.b  (ATEMP)+,(DADDR)+
000001DA 5346              subq.w  #1,TEMP1
000001DC 6EFA              bgt.s   rpt1
000001DE 5345              subq.w  #1,count
000001E0 6EF2              bgt.s   rpt0
```

```
  335    000001E2 4ED0      retrrn  jmp     (RET)
  336                               end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
SYSGLOBALS	ABS	17	00000002

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
ASM_PSUBTOPSUB	REL	29		00000000
ASM_PSUBTOSSUB	REL	195		00000102
ASM_SSUBTOPSUB	REL	71		00000048
ASM_SSUBTOSSUB	REL	114		00000088
ASM_STRLTRIM	REL	276		00000184
ASM_STRRPT	REL	308		000001AC
ASM_STRRTRIM	REL	248		0000015E
ATEMP	AREG	5		00000004
CCR	STREG	0		00000005
COUNT	DREG	12		00000005
CURSLEN	DREG	9		00000002
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DADDR	AREG	3		00000002
DINDEX	DREG	10		00000003
DMAX	DREG	11		00000004
ERROR	REL	124		0000009A
ESCAPECODE	ABS	18	SYSGLOBALS +	FFFFFFFE
INDEX	DREG	13		00000005
L1	REL	122		00000096
L2	REL	126		000000A2
L3	REL	163		000000DE
L4	REL	172		000000EA
L5	REL	182		000000FA
LL1	REL	203		00000110
LL2	REL	239		00000154
LLL1	REL	79		00000056
LLLL1	REL	37		0000000E
LTRIM0	REL	287		00000192
LTRIM1	REL	290		00000196
LTRIM2	REL	293		0000019C
LTRIM3	REL	297		000001A4

NULL	REL	263	00000178
RET	AREG	1	00000000
RETRN	REL	269	00000182
RETRNN	REL	300	000001AA
RETRRN	REL	335	000001E2
RPT0	REL	328	000001D4
RPT1	REL	330	000001D8
RTRIM1	REL	258	0000016E
RTRIM2	REL	266	0000017C
SADDR	AREG	2	00000001
SINDEX	DREG	8	00000001
SLEN	DREG	7	00000000
SP	AREG	0	00000007
SR	STREG	0	00000006
SRCEND	AREG	4	00000003
TEMP1	DREG	14	00000006
TEMP2	DREG	15	00000007
TOPDOWN	REL	179	000000F2
TRANSFER	REL	166	000000E6
USP	STREG	0	00000007

STRG2

Purpose

STRG2 contains assembly language routines to support these Pascal routines:

- STRAPPEND
- STRINSERT
- STRDELETE
- APPEND
- INSERT
- DELETE
- COPY
- CONCAT

Usage

The Compiler emits calls to these routines.

```
 PAGE   1 [2.0] 11/4/82 13:54:38  ASSEMBLY OF STRG2.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1              0000 0000 RET      EQU     A0      return address
    2              0000 0001 SADDR    EQU     A1      source address
    3              0000 0002 DADDR    EQU     A2      destination address
    4              0000 0003 SRCEND   EQU     A3      source end
    5              0000 0004 ATEMP    EQU     A4      temporary A register
    6                        *
    7              0000 0000 SLEN     EQU     D0      source length
    8              0000 0001 SINDEX   EQU     D1      source index
    9              0000 0002 CURSLEN  EQU     D2      current source length
   10              0000 0003 DINDEX   EQU     D3      destination index
   11              0000 0004 DMAX     EQU     D4      destination max
   12              0000 0005 COUNT    EQU     D5
   13              0000 0005 INDEX    EQU     D5      string index
   14              0000 0006 TEMP1    EQU     D6      temporary D register
   15              0000 0007 TEMP2    EQU     D7      temporary D register
   16                        *
   17                                 refa    SYSGLOBALS
   18              FFFF FFFE ESCAPECODE EQU   SYSGLOBALS-2
   19                        *
   20                        *
   21                        *  procedure sappend(var dest: string;
   22                        *                                  src: stringmax);
   23                        *
   24                        *
   25              0000 0000 ASM_SAPPEND   EQU *
   26                                 DEF     ASM_SAPPEND
   27                        *
   28   00000000 205F                movea.l (sp)+,RET
   29   00000002 225F                movea.l (sp)+,SADDR
   30   00000004 245F                movea.l (sp)+,DADDR
   31   00000006 7800                moveq   #0,DMAX
   32   00000008 181F                move.b  (sp)+,DMAX
   33   0000000A 7000                moveq   #0,SLEN
   34   0000000C 1019                move.b  (SADDR)+,SLEN
   35   0000000E 6718                beq.s   retrrnn
   36   00000010 7C00                moveq   #0,TEMP1
   37   00000012 1C12                move.b  (DADDR),TEMP1
   38   00000014 3E06        apend0  move.w  TEMP1,TEMP2
   39   00000016 DE40                add.w   SLEN,TEMP2
   40   00000018 BE44                cmp.w   DMAX,TEMP2
   41   0000001A 6E56                bgt.s   error
   42   0000001C 14C7                move.b  TEMP2,(DADDR)+
   43   0000001E 45F2 6000           lea     0(DADDR,TEMP1.W),DADDR
   44   00000022 14D9        apend1  move.b  (SADDR)+,(DADDR)+
   45   00000024 5340                subq.w  #1,SLEN
   46   00000026 6EFA                bgt.s   apend1
   47   00000028 4ED0        retrrnn jmp     (RET)
   48                        *
   49                        *
   50                        *
   51                        *  procedure insert(src: stringmax;
   52                        *                   var dest: string; index: integer);
   53                        *
   54                        *
   55              0000 002A ASM_INSERT    EQU *
   56                                 DEF     ASM_INSERT
   57                        *
   58   0000002A 205F                movea.l (sp)+,RET
```

```
 PAGE   2 [2.0] 11/4/82 13:54:38  ASSEMBLY OF STRG2.TEXT

   59   0000002C 2A1F                move.l  (sp)+,INDEX
   60   0000002E 245F                movea.l (sp)+,DADDR
   61   00000030 7800                moveq   #0,DMAX
   62   00000032 181F                move.b  (sp)+,DMAX
   63   00000034 225F                movea.l (sp)+,SADDR
   64   00000036 7C00                moveq   #0,TEMP1
   65   00000038 1C12                move.b  (DADDR),TEMP1
   66   0000003A 7000                moveq   #0,SLEN
   67   0000003C 1019                move.b  (SADDR)+,SLEN
   68   0000003E 6730                beq.s   rettrn
   69
   70   00000040 2E06                move.l  TEMP1,TEMP2
   71   00000042 5287                addq.l  #1,TEMP2
   72   00000044 BE85                cmp.l   INDEX,TEMP2
   73   00000046 67CC                beq.s   apend0
   74   00000048 6D28                blt.s   error
   75   0000004A 3E06                move.w  TEMP1,TEMP2
   76   0000004C DE40                add.w   SLEN,TEMP2
   77   0000004E BE44                cmp.w   DMAX,TEMP2
   78   00000050 6E20                bgt.s   error
   79   00000052 14C7                move.b  TEMP2,(DADDR)+
   80   00000054 45F2 6000           lea     0(DADDR,TEMP1.W),DADDR
   81   00000058 49F2 0000           lea     0(DADDR,SLEN.W),ATEMP
   82   0000005C 3E06                move.w  TEMP1,TEMP2
   83   0000005E 9E45                sub.w   INDEX,TEMP2
   84   00000060 1922        insert1 move.b  -(DADDR),-(ATEMP)
   85   00000062 5347                subq.w  #1,TEMP2
   86   00000064 6CFA                bge.s   insert1
   87   00000066 43F1 0000           lea     0(SADDR,SLEN.W),SADDR
   88   0000006A 1921        insert2 move.b  -(SADDR),-(ATEMP)
   89   0000006C 5340                subq.w  #1,SLEN
   90   0000006E 6EFA                bgt.s   insert2
   91   00000070 4ED0        rettrn  jmp     (RET)
   92                        *
   93                        *
   94   00000072 3B7C FFF8 error        move.w      #-8,ESCAPECODE(a5)
                FFFE
   95   00000078 4E4A                   trap        #10
   96                        *
   97                        *
   98                        *
   99                        *  procedure scopy(var dest: string;
  100                        *                  src: stringmax;
  101                        *                  index,length: integer);
  102                        *
  103                        *
  104              0000 007A ASM_SCOPY     EQU *
  105                                 DEF     ASM_SCOPY
  106                        *
  107   0000007A 205F                movea.l (sp)+,RET
  108   0000007C 201F                move.l  (sp)+,SLEN
  109   0000007E 6E0C                bgt.s   copy0
  110   00000080 DFFC 0000                adda.l      #8,sp
                0008
  111   00000086 245F                     movea.l     (sp)+,DADDR
  112   00000088 4212                     clr.b       (DADDR)
  113   0000008A 4ED0                     jmp         (RET)
```

```
PAGE   3 [2.0] 11/4/82 13:54:38  ASSEMBLY OF STRG2.TEXT

  114    0000008C 2A1F       copy0   move.l  (sp)+,INDEX
  115    0000008E 6FE2               ble.s   error
  116
  117    00000090 225F               movea.l (sp)+,SADDR
  118    00000092 245F               movea.l (sp)+,DADDR
  119    00000094 7400               moveq   #0,CURSLEN
  120    00000096 1411               move.b  (SADDR),CURSLEN
  121    00000098 5242               addq.w  #1,CURSLEN
  122    0000009A 9485               sub.l   INDEX,CURSLEN
  123    0000009C 6DD4               blt.s   error
  124
  125    0000009E B042               cmp.w   CURSLEN,SLEN
  126    000000A0 6ED0               bgt.s   error
  127
  128    000000A2 14C0               move.b  SLEN,(DADDR)+
  129    000000A4 43F1 5000          lea     0(SADDR,INDEX.W),SADDR
  130    000000A8 14D9       copy1   move.b  (SADDR)+,(DADDR)+
  131    000000AA 5340               subq.w  #1,SLEN
  132    000000AC 6EFA               bgt.s   copy1
  133    000000AE 4ED0               jmp     (RET)
  134                        *
  135                        *
  136                        *
  137                        *  procedure delete(var dest: string;
  138                        *                   index,length: integer);
  139                        *
  140                        *
  141             0000 00B0 ASM_DELETE    EQU *
  142                               DEF   ASM_DELETE
  143                        *
  144    000000B0 205F               movea.l (sp)+,RET
  145    000000B2 201F               move.l  (sp)+,SLEN
  146    000000B4 6E08               bgt.s   del0
  147    000000B6 DFFC 0000               adda.l     #8,sp
                  0008
  148    000000BC 4ED0                    jmp        (RET)
  149    000000BE 2A1F       del0    move.l  (sp)+,INDEX
  150    000000C0 6FB0               ble.s   error
  151
  152    000000C2 245F               movea.l (sp)+,DADDR
  153    000000C4 7400               moveq   #0,CURSLEN
  154    000000C6 1412               move.b  (DADDR),CURSLEN
  155    000000C8 9480               sub.l   SLEN,CURSLEN
  156    000000CA 6DA6               blt.s   error
  157
  158    000000CC 3C02               move.w  CURSLEN,TEMP1
  159    000000CE 5246               addq    #1,TEMP1
  160    000000D0 9C45               sub.w   INDEX,TEMP1
  161    000000D2 6D9E               blt.s   error
  162
  163    000000D4 1482               move.b  CURSLEN,(DADDR)
  164    000000D6 45F2 5000          lea     0(DADDR,INDEX.W),DADDR
  165    000000DA 43F2 0000          lea     0(DADDR,SLEN.W),SADDR
  166    000000DE 14D9       del1    move.b  (SADDR)+,(DADDR)+
  167    000000E0 5346               subq.w  #1,TEMP1
  168    000000E2 6EFA               bgt.s   del1
  169    000000E4 4ED0               jmp     (RET)
```

```
PAGE   4 [2.0] 11/4/82 13:54:38  ASSEMBLY OF STRG2.TEXT

  170
  171                               end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
SYSGLOBALS	ABS	17	00000002

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
APEND0	REL	38		00000014
APEND1	REL	44		00000022
ASM_DELETE	REL	141		000000B0
ASM_INSERT	REL	55		0000002A
ASM_SAPPEND	REL	25		00000000
ASM_SCOPY	REL	104		0000007A
ATEMP	AREG	5		00000004
CCR	STREG	0		00000005
COPY0	REL	114		0000008C
COPY1	REL	130		000000A8
COUNT	DREG	12		00000005
CURSLEN	DREG	9		00000002
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007
DADDR	AREG	3		00000002
DEL0	REL	149		000000BE
DEL1	REL	166		000000DE
DINDEX	DREG	10		00000003
DMAX	DREG	11		00000004
ERROR	REL	94		00000072
ESCAPECODE	ABS	18	SYSGLOBALS +	FFFFFFFE
INDEX	DREG	13		00000005
INSERT1	REL	84		00000060
INSERT2	REL	88		0000006A
RET	AREG	1		00000000
RETRRNN	REL	47		00000028
RETTRN	REL	91		00000070
SADDR	AREG	2		00000001
SINDEX	DREG	8		00000001
SLEN	DREG	7		00000000
SP	AREG	0		00000007
SR	STREG	0		00000006

SRCEND	AREG	4		00000003
TEMP1	DREG	14		00000006
TEMP2	DREG	15		00000007
USP	STREG	0		00000007

STRINT

Purpose

STRINT implements the conversion of an integer to a string of digits.

Usage

The Compiler emits a call to this procedure for STRWRITE of an integer.

```
 PAGE   1 [2.0] 11/4/82 13:14:52  ASSEMBLY OF STRINT.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1                                   def fs_fwritestrint
    2                                   refa sysglobals
    3
    4               FFFF FFEA ioresult          equ sysglobals-22
    5               0000 001C istrovfl          equ 28
    6
    7               0000 0000 sign              equ d0
    8               0000 0001 digits            equ d1
    9
   10               0000 0002 int               equ d2
   11               0000 0003 high              equ d3
   12               0000 0004 extend            equ d4
   13               0000 0005 ten               equ d5
   14               0000 0006 zero              equ d6
   15               0000 0007 fieldwidth        equ d7
   16
   17               0000 0000 maxstrlen         equ d0
   18
   19               0000 0002 intlen            equ d2
   20               0000 0003 strlen            equ d3
   21               0000 0004 temp              equ d4
   22               0000 0005 newindex          equ d5
   23               0000 0006 index             equ d6
   24
   25               0000 0000 return            equ a0
   26               0000 0001 straddr           equ a1
   27               0000 0002 dindex            equ a2
   28               0000 0003 sindex            equ a3
   29               0000 0004 aindex            equ a4
   30
   31      00000000 84C5      10       divu    ten,int
   32      00000002 4842               swap    int
   33      00000004 D446               add.w   zero,int
   34      00000006 1502               move.b  int,-(dindex)
   35      00000008 5241               addq    #1,digits
   36      0000000A 4242               clr.w   int
   37      0000000C 4842               swap    int
   38               0000 000E get_digits       equ *
   39      0000000E B485               cmp.l   ten,int
   40      00000010 64EE               bcc.s   10
   41      00000012 D446               add.w   zero,int
   42      00000014 1502               move.b  int,-(dindex)
   43      00000016 5241               addq    #1,digits
   44      00000018 4E75               rts
   45
   46               0000 001A fs_fwritestrint  equ *
   47      0000001A 205F               movea.l (sp)+,return
   48      0000001C 3E1F               move    (sp)+,fieldwidth        integer is to be right justified
   49      0000001E 241F               move.l  (sp)+,int               integer to be converted
   50      00000020 285F               movea.l (sp)+,aindex            address of index into string
   51      00000022 225F               movea.l (sp)+,straddr           address of string
   52      00000024 7000               moveq   #0,maxstrlen
   53      00000026 101F               move.b  (sp)+,maxstrlen         dimensioned size of string
   54
   55      00000028 4E56 FFF4          link    a6,#-12                 allocate space for local string
   56      0000002C 1E80               move.b  maxstrlen,(sp)          save maxstrlen in local string
   57      0000002E 45D6               lea     (a6),dindex             index to tail of local string
   58      00000030 7200               moveq   #0,digits
```

```
 PAGE   2 [2.0] 11/4/82 13:14:52  ASSEMBLY OF STRINT.TEXT

   59      00000032 7A0A               moveq   #10,ten
   60      00000034 7C30               moveq   #'0',zero
   61
   62      00000036 4A82               tst.l   int
   63      00000038 5DC0               slt     sign            remember whether negative
   64      0000003A 6C02               bge.s   positive
   65      0000003C 4482               neg.l   int
   66      0000003E 7600      positive moveq   #0,high
   67      00000040 B4BC 0001          cmp.l   #100000,int
           00086A0
   68      00000046 6512               bcs.s   small           unsigned integer is less than 100000
   69
   70                        * divide integer by 100000, save high part, keep low part in integer:
   71
   72      00000048 E28A               lsr.l   #1,int          divide by 2
   73      0000004A 40C4               move    sr,extend       remainder in extend bit
   74      0000004C 84FC C350          divu    #50000,int      divide by 50000
   75      00000050 3602               move.w  int,high        quotient is high part
   76      00000052 4242               clr.w   int             zap quotient
   77      00000054 4842               swap    int             get remainder
   78      00000056 44C4               move    extend,ccr      reconstruct full remainder
   79      00000058 E392               roxl.l  #1,int          integer is now mod 100000
   80
   81      0000005A 61B2      small    bsr.s   get_digits      extract low order digits
   82
   83      0000005C 4A43               tst.w   high            check for any high order digits
   84      0000005E 6710               beq.s   finish
   85      00000060 B27C 0005 l1       cmp     #5,digits       there should be at least 5 digits
   86      00000064 6706               beq.s   l2
   87      00000066 1506               move.b  zero,-(dindex)
   88      00000068 5241               addq    #1,digits
   89      0000006A 60F4               bra.s   l1
   90      0000006C 3403      l2       move    high,int
   91      0000006E 619E               bsr.s   get_digits      get high order digits
   92
   93               0000 0070 finish   equ *
   94
   95      00000070 2C14               move.l  (aindex),index
   96      00000072 6F60               ble.s   error                   error if index < 1
   97
   98      00000074 7600               moveq   #0,strlen
   99      00000076 1611               move.b  (straddr),strlen        current length of string
  100
  101      00000078 2803               move.l  strlen,temp
  102      0000007A 5284               addq.l  #1,temp
  103      0000007C BC84               cmp.l   temp,index
  104      0000007E 6E54               bgt.s   error           error if index > strlen(s)+1
  105
  106      00000080 3401               move    digits,intlen   compute minimum size of new integer
  107      00000082 4A00               tst.b   sign
  108      00000084 6702               beq.s   nosign
  109      00000086 5242               addq    #1,intlen       add 1 for minus sign
  110               0000 0088 nosign   equ *
  111
  112      00000088 4A47               tst     fieldwidth      field width is 12 if passed negative
  113      0000008A 6C02               bge.s   nodefault
  114      0000008C 7E0C               moveq   #12,fieldwidth
```

```
 115              0000 008E nodefault equ *
 116
 117     0000008E BE42              cmp   intlen,fieldwidth       fieldwidth is at least size of integer
 118     00000090 6C02              bge.s   bigger
 119     00000092 3E02              move    intlen,fieldwidth
 120              0000 0094 bigger  equ *
 121
 122     00000094 2A06              move.l  index,newindex          compute new index
 123     00000096 DA47              add     fieldwidth,newindex
 124
 125     00000098 3805              move    newindex,temp           string length is maximum of
 126     0000009A 5344              subq    #1,temp                         old length, newindex-1
 127     0000009C B644              cmp     temp,strlen
 128     0000009E 6C02              bge.s   longer
 129     000000A0 3604              move    temp,strlen
 130              0000 00A2 longer  equ     *
 131
 132     000000A2 4244              clr     temp
 133     000000A4 1817              move.b  (sp),temp               retrieve strmax
 134     000000A6 B644              cmp     temp,strlen
 135     000000A8 6E2A              bgt.s   error                   error if new length > strmax(s)
 136
 137     000000AA 1283              move.b  strlen,(straddr)        set new string length
 138     000000AC 2885              move.l  newindex,(aindex)               set new index
 139
 140     000000AE 47F1 6000         lea     0(straddr,index),sindex
 141
 142     000000B2 9E42              sub     intlen,fieldwidth
 143     000000B4 6708              beq.s   nospaces
 144     000000B6 16FC 0020 space   move.b  #' ',(sindex)+
 145     000000BA 5347              subq    #1,fieldwidth
 146     000000BC 6EF8              bgt.s   space
 147              0000 00BE nospaces equ *
 148
 149     000000BE 4A00              tst.b  sign
 150     000000C0 6704              beq.s   nosgn
 151     000000C2 16FC 002D         move.b  #'-',(sindex)+
 152              0000 00C6 nosgn   equ *
 153
 154     000000C6 16DA      digit   move.b  (dindex)+,(sindex)+
 155     000000C8 5341              subq    #1,digits
 156     000000CA 6EFA              bgt.s   digit
 157
 158     000000CC 42AD FFEA         clr.l   ioresult(a5)
 159     000000D0 4E5E      ok      unlk    a6
 160     000000D2 4ED0              jmp     (return)
 161
 162     000000D4 701C      error   moveq   #istrovfl,d0
 163     000000D6 2B40 FFEA         move.l  d0,ioresult(a5)
 164     000000DA 60F4              bra.s   ok
 165
 166                                nosyms
 167                                end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

SYSDEF

Purpose

SYSDEF provides the symbol table for SYSTEM_P.

Usage

SYSDEF is accessed by the loader.

Notes

(This file was generated programatically!)

```
PAGE   1 [2.0] 11/4/82 13:16:52  ASSEMBLY OF SYSDEF.TEXT

PASS 1 COMPLETE. ERRORS: 0
    1    00000000                    rorg 0
    2
    3                                def  sysdeftable
    4
    5              0000 0000 sysdeftable equ *              module descriptor:
    6    00000000 0000 0000          dc.l    0             link = NIL
    7    00000004 00                 dc.b    0             patchmod = false
    8    00000006 0000 0028          dc.l    defaddr       address of DEF table
    9    0000000A 0000 06DC          dc.l    endd-defaddr  defsize
   10    0000000E 01                 dc.b    1             resolved = true
   11    00000010 0000 0000          dc.l    0             startaddress
   12    00000014 0B                 dc.b    11,'SYSDEFTABLE      '     progname
   12    00000015 53
   12    00000016 59
   12    00000017 53
   12    00000018 44
   12    00000019 45
   12    0000001A 46
   12    0000001B 54
   12    0000001C 41
   12    0000001D 42
   12    0000001E 4C
   12    0000001F 45
   12    00000020 20
   12    00000021 20
   12    00000022 20
   12    00000023 20
   12    00000024 20
   13    00000025 00         x       dc.b    0,1
   13    00000026 01
   14
   15              0000 1206 GVR      equ     $1206          abs, sint, value extend, 6 bytes
   16
   17              0000 0028 defaddr  equ *
   18
   19                                 include SYSTABLE
   20                            REFR  ALPHALIST
   21    00000028 09             DC.B  9,'ALPHALIST'
   21    00000029 41
   21    0000002A 4C
   21    0000002B 50
   21    0000002C 48
   21    0000002D 41
   21    0000002E 4C
   21    0000002F 49
   21    00000030 53
   21    00000031 54
   22    00000032 1206           DC.W  GVR
   23    00000034 0000 0000      DC.L  ALPHALIST
   24                            REFR  ASM_ASM
   25    00000038 07             DC.B  7,'ASM_ASM'
   25    00000039 41
   25    0000003A 53
   25    0000003B 4D
   25    0000003C 5F
   25    0000003D 41
   25    0000003E 53
   25    0000003F 4D
```

```
PAGE   2 [2.0] 11/4/82 13:16:52  ASSEMBLY OF SYSDEF.TEXT

   26    00000040 1206           DC.W  GVR
   27    00000042 0000 0000      DC.L  ASM_ASM
   28                            REFR  ASM_ASSIGN
   29    00000046 0A             DC.B  10,'ASM_ASSIGN'
   29    00000047 41
   29    00000048 53
   29    00000049 4D
   29    0000004A 5F
   29    0000004B 41
   29    0000004C 53
   29    0000004D 53
   29    0000004E 49
   29    0000004F 47
   29    00000050 4E
   30    00000052 1206           DC.W  GVR
   31    00000054 0000 0000      DC.L  ASM_ASSIGN
   32                            REFR  ASM_CI_SWITCH
   33    00000058 0D             DC.B  13,'ASM_CI_SWITCH'
   33    00000059 41
   33    0000005A 53
   33    0000005B 4D
   33    0000005C 5F
   33    0000005D 43
   33    0000005E 49
   33    0000005F 5F
   33    00000060 53
   33    00000061 57
   33    00000062 49
   33    00000063 54
   33    00000064 43
   33    00000065 48
   34    00000066 1206           DC.W  GVR
   35    00000068 0000 0000      DC.L  ASM_CI_SWITCH
   36                            REFR  ASM_CPYMSG
   37    0000006C 0A             DC.B  10,'ASM_CPYMSG'
   37    0000006D 41
   37    0000006E 53
   37    0000006F 4D
   37    00000070 5F
   37    00000071 43
   37    00000072 50
   37    00000073 59
   37    00000074 4D
   37    00000075 53
   37    00000076 47
   38    00000078 1206           DC.W  GVR
   39    0000007A 0000 0000      DC.L  ASM_CPYMSG
   40                            REFR  ASM_DIFFERENCE
   41    0000007E 0E             DC.B  14,'ASM_DIFFERENCE'
   41    0000007F 41
   41    00000080 53
   41    00000081 4D
   41    00000082 5F
   41    00000083 44
   41    00000084 49
   41    00000085 46
   41    00000086 46
```

```
41    00000087 45
41    00000088 52
41    00000089 45
41    0000008A 4E
41    0000008B 43
41    0000008C 45
42    0000008E 1206          DC.W  GVR
43    00000090 0000 0000     DC.L  ASM_DIFFERENCE
44                           REFR  ASM_DIV
45    00000094 07            DC.B  7,'ASM_DIV'
45    00000095 41
45    00000096 53
45    00000097 4D
45    00000098 5F
45    00000099 44
45    0000009A 49
45    0000009B 56
46    0000009C 1206          DC.W  GVR
47    0000009E 0000 0000     DC.L  ASM_DIV
48                           REFR  ASM_EQUAL
49    000000A2 09            DC.B  9,'ASM_EQUAL'
49    000000A3 41
49    000000A4 53
49    000000A5 4D
49    000000A6 5F
49    000000A7 45
49    000000A8 51
49    000000A9 55
49    000000AA 41
49    000000AB 4C
50    000000AC 1206          DC.W  GVR
51    000000AE 0000 0000     DC.L  ASM_EQUAL
52                           REFR  ASM_ERRMSG
53    000000B2 0A            DC.B  10,'ASM_ERRMSG'
53    000000B3 41
53    000000B4 53
53    000000B5 4D
53    000000B6 5F
53    000000B7 45
53    000000B8 52
53    000000B9 52
53    000000BA 4D
53    000000BB 53
53    000000BC 47
54    000000BE 1206          DC.W  GVR
55    000000C0 0000 0000     DC.L  ASM_ERRMSG
56                           REFR  ASM_FASTMOVE
57    000000C4 0C            DC.B  12,'ASM_FASTMOVE'
57    000000C5 41
57    000000C6 53
57    000000C7 4D
57    000000C8 5F
57    000000C9 46
57    000000CA 41
57    000000CB 53
57    000000CC 54
57    000000CD 4D
```



```
57    000000CE 4F
57    000000CF 56
57    000000D0 45
58    000000D2 1206          DC.W  GVR
59    000000D4 0000 0000     DC.L  ASM_FASTMOVE
60                           REFR  ASM_FINDROMS
61    000000D8 0C            DC.B  12,'ASM_FINDROMS'
61    000000D9 41
61    000000DA 53
61    000000DB 4D
61    000000DC 5F
61    000000DD 46
61    000000DE 49
61    000000DF 4E
61    000000E0 44
61    000000E1 52
61    000000E2 4F
61    000000E3 4D
61    000000E4 53
62    000000E6 1206          DC.W  GVR
63    000000E8 0000 0000     DC.L  ASM_FINDROMS
64                           REFR  ASM_FLPYINIT
65    000000EC 0C            DC.B  12,'ASM_FLPYINIT'
65    000000ED 41
65    000000EE 53
65    000000EF 4D
65    000000F0 5F
65    000000F1 46
65    000000F2 4C
65    000000F3 50
65    000000F4 59
65    000000F5 49
65    000000F6 4E
65    000000F7 49
65    000000F8 54
66    000000FA 1206          DC.W  GVR
67    000000FC 0000 0000     DC.L  ASM_FLPYINIT
68                           REFR  ASM_FLPYMREAD
69    00000100 0D            DC.B  13,'ASM_FLPYMREAD'
69    00000101 41
69    00000102 53
69    00000103 4D
69    00000104 5F
69    00000105 46
69    00000106 4C
69    00000107 50
69    00000108 59
69    00000109 4D
69    0000010A 52
69    0000010B 45
69    0000010C 41
69    0000010D 44
70    0000010E 1206          DC.W  GVR
71    00000110 0000 0000     DC.L  ASM_FLPYMREAD
72                           REFR  ASM_FLPYMWRITE
73    00000114 0E            DC.B  14,'ASM_FLPYMWRITE'
73    00000115 41
```

```
  73    00000116 53
  73    00000117 4D
  73    00000118 5F
  73    00000119 46
  73    0000011A 4C
  73    0000011B 50
  73    0000011C 59
  73    0000011D 4D
  73    0000011E 57
  73    0000011F 52
  73    00000120 49
  73    00000121 54
  73    00000122 45
  74    00000124 1206          DC.W  GVR
  75    00000126 0000 0000     DC.L  ASM_FLPYMWRITE
  76                           REFR  ASM_FLPYREAD
  77    0000012A 0C            DC.B  12,'ASM_FLPYREAD'
  77    0000012B 41
  77    0000012C 53
  77    0000012D 4D
  77    0000012E 5F
  77    0000012F 46
  77    00000130 4C
  77    00000131 50
  77    00000132 59
  77    00000133 52
  77    00000134 45
  77    00000135 41
  77    00000136 44
  78    00000138 1206          DC.W  GVR
  79    0000013A 0000 0000     DC.L  ASM_FLPYREAD
  80                           REFR  ASM_FLPY_WRT
  81    0000013E 0C            DC.B  12,'ASM_FLPY_WRT'
  81    0000013F 41
  81    00000140 53
  81    00000141 4D
  81    00000142 5F
  81    00000143 46
  81    00000144 4C
  81    00000145 50
  81    00000146 59
  81    00000147 5F
  81    00000148 57
  81    00000149 52
  81    0000014A 54
  82    0000014C 1206          DC.W  GVR
  83    0000014E 0000 0000     DC.L  ASM_FLPY_WRT
  84                           REFR  ASM_F_PWR_ON
  85    00000152 0C            DC.B  12,'ASM_F_PWR_ON'
  85    00000153 41
  85    00000154 53
  85    00000155 4D
  85    00000156 5F
  85    00000157 46
  85    00000158 5F
  85    00000159 50
  85    0000015A 57
```



```
  85    0000015B 52
  85    0000015C 5F
  85    0000015D 4F
  85    0000015E 4E
  86    00000160 1206          DC.W  GVR
  87    00000162 0000 0000     DC.L  ASM_F_PWR_ON
  88                           REFR  ASM_IN
  89    00000166 06            DC.B  6,'ASM_IN'
  89    00000167 41
  89    00000168 53
  89    00000169 4D
  89    0000016A 5F
  89    0000016B 49
  89    0000016C 4E
  90    0000016E 1206          DC.W  GVR
  91    00000170 0000 0000     DC.L  ASM_IN
  92                           REFR  ASM_INCLUSION
  93    00000174 0D            DC.B  13,'ASM_INCLUSION'
  93    00000175 41
  93    00000176 53
  93    00000177 4D
  93    00000178 5F
  93    00000179 49
  93    0000017A 4E
  93    0000017B 43
  93    0000017C 4C
  93    0000017D 55
  93    0000017E 53
  93    0000017F 49
  93    00000180 4F
  93    00000181 4E
  94    00000182 1206          DC.W  GVR
  95    00000184 0000 0000     DC.L  ASM_INCLUSION
  96                           REFR  ASM_INTERSECT
  97    00000188 0D            DC.B  13,'ASM_INTERSECT'
  97    00000189 41
  97    0000018A 53
  97    0000018B 4D
  97    0000018C 5F
  97    0000018D 49
  97    0000018E 4E
  97    0000018F 54
  97    00000190 45
  97    00000191 52
  97    00000192 53
  97    00000193 45
  97    00000194 43
  97    00000195 54
  98    00000196 1206          DC.W  GVR
  99    00000198 0000 0000     DC.L  ASM_INTERSECT
 100                           REFR  ASM_MEMAVAIL
 101    0000019C 0C            DC.B  12,'ASM_MEMAVAIL'
 101    0000019D 41
 101    0000019E 53
 101    0000019F 4D
 101    000001A0 5F
 101    000001A1 4D
```

```
101     000001A2 45
101     000001A3 4D
101     000001A4 41
101     000001A5 56
101     000001A6 41
101     000001A7 49
101     000001A8 4C
102     000001AA 1206           DC.W  GVR
103     000001AC 0000 0000      DC.L  ASM_MEMAVAIL
104                             REFR  ASM_MOD
105     000001B0 07             DC.B  7,'ASM_MOD'
105     000001B1 41
105     000001B2 53
105     000001B3 4D
105     000001B4 5F
105     000001B5 4D
105     000001B6 4F
105     000001B7 44
106     000001B8 1206           DC.W  GVR
107     000001BA 0000 0000      DC.L  ASM_MOD
108                             REFR  ASM_MOVEL
109     000001BE 09             DC.B  9,'ASM_MOVEL'
109     000001BF 41
109     000001C0 53
109     000001C1 4D
109     000001C2 5F
109     000001C3 4D
109     000001C4 4F
109     000001C5 56
109     000001C6 45
109     000001C7 4C
110     000001C8 1206           DC.W  GVR
111     000001CA 0000 0000      DC.L  ASM_MOVEL
112                             REFR  ASM_MOVELEFT
113     000001CE 0C             DC.B  12,'ASM_MOVELEFT'
113     000001CF 41
113     000001D0 53
113     000001D1 4D
113     000001D2 5F
113     000001D3 4D
113     000001D4 4F
113     000001D5 56
113     000001D6 45
113     000001D7 4C
113     000001D8 45
113     000001D9 46
113     000001DA 54
114     000001DC 1206           DC.W  GVR
115     000001DE 0000 0000      DC.L  ASM_MOVELEFT
116                             REFR  ASM_MOVER
117     000001E2 09             DC.B  9,'ASM_MOVER'
117     000001E3 41
117     000001E4 53
117     000001E5 4D
117     000001E6 5F
117     000001E7 4D
117     000001E8 4F
```



```
117     000001E9 56
117     000001EA 45
117     000001EB 52
118     000001EC 1206           DC.W  GVR
119     000001EE 0000 0000      DC.L  ASM_MOVER
120                             REFR  ASM_MOVERIGHT
121     000001F2 0D             DC.B  13,'ASM_MOVERIGHT'
121     000001F3 41
121     000001F4 53
121     000001F5 4D
121     000001F6 5F
121     000001F7 4D
121     000001F8 4F
121     000001F9 56
121     000001FA 45
121     000001FB 52
121     000001FC 49
121     000001FD 47
121     000001FE 48
121     000001FF 54
122     00000200 1206           DC.W  GVR
123     00000202 0000 0000      DC.L  ASM_MOVERIGHT
124                             REFR  ASM_MPY
125     00000206 07             DC.B  7,'ASM_MPY'
125     00000207 41
125     00000208 53
125     00000209 4D
125     0000020A 5F
125     0000020B 4D
125     0000020C 50
125     0000020D 59
126     0000020E 1206           DC.W  GVR
127     00000210 0000 0000      DC.L  ASM_MPY
128                             REFR  ASM_NEQUAL
129     00000214 0A             DC.B  10,'ASM_NEQUAL'
129     00000215 41
129     00000216 53
129     00000217 4D
129     00000218 5F
129     00000219 4E
129     0000021A 45
129     0000021B 51
129     0000021C 55
129     0000021D 41
129     0000021E 4C
130     00000220 1206           DC.W  GVR
131     00000222 0000 0000      DC.L  ASM_NEQUAL
132                             REFR  ASM_NEWBYTES
133     00000226 0C             DC.B  12,'ASM_NEWBYTES'
133     00000227 41
133     00000228 53
133     00000229 4D
133     0000022A 5F
133     0000022B 4E
133     0000022C 45
133     0000022D 57
133     0000022E 42
```

```
133    0000022F 59
133    00000230 54
133    00000231 45
133    00000232 53
134    00000234 1206          DC.W  GVR
135    00000236 0000 0000     DC.L  ASM_NEWBYTES
136                           REFR  ASM_NEWWORDS
137    0000023A 0C            DC.B  12,'ASM_NEWWORDS'
137    0000023B 41
137    0000023C 53
137    0000023D 4D
137    0000023E 5F
137    0000023F 4E
137    00000240 45
137    00000241 57
137    00000242 57
137    00000243 4F
137    00000244 52
137    00000245 44
137    00000246 53
138    00000248 1206          DC.W  GVR
139    0000024A 0000 0000     DC.L  ASM_NEWWORDS
140                           REFR  ASM_POS
141    0000024E 07            DC.B  7,'ASM_POS'
141    0000024F 41
141    00000250 53
141    00000251 4D
141    00000252 5F
141    00000253 50
141    00000254 4F
141    00000255 53
142    00000256 1206          DC.W  GVR
143    00000258 0000 0000     DC.L  ASM_POS
144                           REFR  ASM_POWERUP
145    0000025C 0B            DC.B  11,'ASM_POWERUP'
145    0000025D 41
145    0000025E 53
145    0000025F 4D
145    00000260 5F
145    00000261 50
145    00000262 4F
145    00000263 57
145    00000264 45
145    00000265 52
145    00000266 55
145    00000267 50
146    00000268 1206          DC.W  GVR
147    0000026A 0000 0000     DC.L  ASM_POWERUP
148                           REFR  ASM_RMOVEL
149    0000026E 0A            DC.B  10,'ASM_RMOVEL'
149    0000026F 41
149    00000270 53
149    00000271 4D
149    00000272 5F
149    00000273 52
149    00000274 4D
149    00000275 4F
```



```
149    00000276 56
149    00000277 45
149    00000278 4C
150    0000027A 1206          DC.W  GVR
151    0000027C 0000 0000     DC.L  ASM_RMOVEL
152                           REFR  ASM_RMOVER
153    00000280 0A            DC.B  10,'ASM_RMOVER'
153    00000281 41
153    00000282 53
153    00000283 4D
153    00000284 5F
153    00000285 52
153    00000286 4D
153    00000287 4F
153    00000288 56
153    00000289 45
153    0000028A 52
154    0000028C 1206          DC.W  GVR
155    0000028E 0000 0000     DC.L  ASM_RMOVER
156                           REFR  ASM_UNION
157    00000292 09            DC.B  9,'ASM_UNION'
157    00000293 41
157    00000294 53
157    00000295 4D
157    00000296 5F
157    00000297 55
157    00000298 4E
157    00000299 49
157    0000029A 4F
157    0000029B 4E
158    0000029C 1206          DC.W  GVR
159    0000029E 0000 0000     DC.L  ASM_UNION
160                           REFR  ASM_USERPROGRAM
161    000002A2 0F            DC.B  15,'ASM_USERPROGRAM'
161    000002A3 41
161    000002A4 53
161    000002A5 4D
161    000002A6 5F
161    000002A7 55
161    000002A8 53
161    000002A9 45
161    000002AA 52
161    000002AB 50
161    000002AC 52
161    000002AD 4F
161    000002AE 47
161    000002AF 52
161    000002B0 41
161    000002B1 4D
162    000002B2 1206          DC.W  GVR
163    000002B4 0000 0000     DC.L  ASM_USERPROGRAM
164                           REFR  BOOTDAMMODULE
165    000002B8 0D            DC.B  13,'BOOTDAMMODULE'
165    000002B9 42
165    000002BA 4F
165    000002BB 4F
165    000002BC 54
```

```
 165    000002BD 44
 165    000002BE 41
 165    000002BF 4D
 165    000002C0 4D
 165    000002C1 4F
 165    000002C2 44
 165    000002C3 55
 165    000002C4 4C
 165    000002C5 45
 166    000002C6 1206          DC.W  GVR
 167    000002C8 0000 0000     DC.L  BOOTDAMMODULE
 168                           REFR  BOOTDAMMODULE_BOOTDAM
 169    000002CC 15            DC.B  21,'BOOTDAMMODULE_BOOTDAM'
 169    000002CD 42
 169    000002CE 4F
 169    000002CF 4F
 169    000002D0 54
 169    000002D1 44
 169    000002D2 41
 169    000002D3 4D
 169    000002D4 4D
 169    000002D5 4F
 169    000002D6 44
 169    000002D7 55
 169    000002D8 4C
 169    000002D9 45
 169    000002DA 5F
 169    000002DB 42
 169    000002DC 4F
 169    000002DD 4F
 169    000002DE 54
 169    000002DF 44
 169    000002E0 41
 169    000002E1 4D
 170    000002E2 1206          DC.W  GVR
 171    000002E4 0000 0000     DC.L  BOOTDAMMODULE_BOOTDAM
 172                           REFR  BOOTDAMMODULE_BOOTDAMMODULE
 173    000002E8 1B            DC.B  27,'BOOTDAMMODULE_BOOTDAMMODULE'
 173    000002E9 42
 173    000002EA 4F
 173    000002EB 4F
 173    000002EC 54
 173    000002ED 44
 173    000002EE 41
 173    000002EF 4D
 173    000002F0 4D
 173    000002F1 4F
 173    000002F2 44
 173    000002F3 55
 173    000002F4 4C
 173    000002F5 45
 173    000002F6 5F
 173    000002F7 42
 173    000002F8 4F
 173    000002F9 4F
 173    000002FA 54
 173    000002FB 44
```



```
 173    000002FC 41
 173    000002FD 4D
 173    000002FE 4D
 173    000002FF 4F
 173    00000300 44
 173    00000301 55
 173    00000302 4C
 173    00000303 45
 174    00000304 1206          DC.W  GVR
 175    00000306 0000 0000     DC.L  BOOTDAMMODULE_BOOTDAMMODULE
 176                           REFR  BOOTDAMMODULE_BOOTNODE
 177    0000030A 16            DC.B  22,'BOOTDAMMODULE_BOOTNODE'
 177    0000030B 42
 177    0000030C 4F
 177    0000030D 4F
 177    0000030E 54
 177    0000030F 44
 177    00000310 41
 177    00000311 4D
 177    00000312 4D
 177    00000313 4F
 177    00000314 44
 177    00000315 55
 177    00000316 4C
 177    00000317 45
 177    00000318 5F
 177    00000319 42
 177    0000031A 4F
 177    0000031B 4F
 177    0000031C 54
 177    0000031D 4E
 177    0000031E 4F
 177    0000031F 44
 177    00000320 45
 178    00000322 1206          DC.W  GVR
 179    00000324 0000 0000     DC.L  BOOTDAMMODULE_BOOTNODE
 180                           REFR  BOOTDAMMODULE_BOOTTM
 181    00000328 14            DC.B  20,'BOOTDAMMODULE_BOOTTM'
 181    00000329 42
 181    0000032A 4F
 181    0000032B 4F
 181    0000032C 54
 181    0000032D 44
 181    0000032E 41
 181    0000032F 4D
 181    00000330 4D
 181    00000331 4F
 181    00000332 44
 181    00000333 55
 181    00000334 4C
 181    00000335 45
 181    00000336 5F
 181    00000337 42
 181    00000338 4F
 181    00000339 4F
 181    0000033A 54
 181    0000033B 54
```

```
181     0000033C 4D
182     0000033E 1206           DC.W  GVR
183     00000340 0000 0000      DC.L  BOOTDAMMODULE_BOOTTM
184                             REFR  BOOTDAMMODULE_INITBOOTDAM
185     00000344 19             DC.B  25,'BOOTDAMMODULE_INITBOOTDAM'
185     00000345 42
185     00000346 4F
185     00000347 4F
185     00000348 54
185     00000349 44
185     0000034A 41
185     0000034B 4D
185     0000034C 4D
185     0000034D 4F
185     0000034E 44
185     0000034F 55
185     00000350 4C
185     00000351 45
185     00000352 5F
185     00000353 49
185     00000354 4E
185     00000355 49
185     00000356 54
185     00000357 42
185     00000358 4F
185     00000359 4F
185     0000035A 54
185     0000035B 44
185     0000035C 41
185     0000035D 4D
186     0000035E 1206           DC.W  GVR
187     00000360 0000 0000      DC.L  BOOTDAMMODULE_INITBOOTDAM
188                             REFR  BOOTDAMMODULE_SRMNODE
189     00000364 15             DC.B  21,'BOOTDAMMODULE_SRMNODE'
189     00000365 42
189     00000366 4F
189     00000367 4F
189     00000368 54
189     00000369 44
189     0000036A 41
189     0000036B 4D
189     0000036C 4D
189     0000036D 4F
189     0000036E 44
189     0000036F 55
189     00000370 4C
189     00000371 45
189     00000372 5F
189     00000373 53
189     00000374 52
189     00000375 4D
189     00000376 4E
189     00000377 4F
189     00000378 44
189     00000379 45
190     0000037A 1206           DC.W  GVR
191     0000037C 0000 0000      DC.L  BOOTDAMMODULE_SRMNODE
```



```
192                             REFR  BOOTDAMMODULE__BASE
193     00000380 13             DC.B  19,'BOOTDAMMODULE__BASE'
193     00000381 42
193     00000382 4F
193     00000383 4F
193     00000384 54
193     00000385 44
193     00000386 41
193     00000387 4D
193     00000388 4D
193     00000389 4F
193     0000038A 44
193     0000038B 55
193     0000038C 4C
193     0000038D 45
193     0000038E 5F
193     0000038F 5F
193     00000390 42
193     00000391 41
193     00000392 53
193     00000393 45
194     00000394 1206           DC.W  GVR
195     00000396 0000 0000      DC.L  BOOTDAMMODULE__BASE
196                             REFR  BOOT_FINDFILE
197     0000039A 0D             DC.B  13,'BOOT_FINDFILE'
197     0000039B 42
197     0000039C 4F
197     0000039D 4F
197     0000039E 54
197     0000039F 5F
197     000003A0 46
197     000003A1 49
197     000003A2 4E
197     000003A3 44
197     000003A4 46
197     000003A5 49
197     000003A6 4C
197     000003A7 45
198     000003A8 1206           DC.W  GVR
199     000003AA 0000 0000      DC.L  BOOT_FINDFILE
200                             REFR  BOOT_LIFHEAD
201     000003AE 0C             DC.B  12,'BOOT_LIFHEAD'
201     000003AF 42
201     000003B0 4F
201     000003B1 4F
201     000003B2 54
201     000003B3 5F
201     000003B4 4C
201     000003B5 49
201     000003B6 46
201     000003B7 48
201     000003B8 45
201     000003B9 41
201     000003BA 44
202     000003BC 1206           DC.W  GVR
203     000003BE 0000 0000      DC.L  BOOT_LIFHEAD
204                             REFR  BOOT_MFCLOSE
```

```
205    000003C2 0C           DC.B  12,'BOOT_MFCLOSE'
205    000003C3 42
205    000003C4 4F
205    000003C5 4F
205    000003C6 54
205    000003C7 5F
205    000003C8 4D
205    000003C9 46
205    000003CA 43
205    000003CB 4C
205    000003CC 4F
205    000003CD 53
205    000003CE 45
206    000003D0 1206         DC.W  GVR
207    000003D2 0000 0000    DC.L  BOOT_MFCLOSE
208                          REFR  BOOT_MFOPEN
209    000003D6 0B           DC.B  11,'BOOT_MFOPEN'
209    000003D7 42
209    000003D8 4F
209    000003D9 4F
209    000003DA 54
209    000003DB 5F
209    000003DC 4D
209    000003DD 46
209    000003DE 4F
209    000003DF 50
209    000003E0 45
209    000003E1 4E
210    000003E2 1206         DC.W  GVR
211    000003E4 0000 0000    DC.L  BOOT_MFOPEN
212                          REFR  BOOT_MINIT
213    000003E8 0A           DC.B  10,'BOOT_MINIT'
213    000003E9 42
213    000003EA 4F
213    000003EB 4F
213    000003EC 54
213    000003ED 5F
213    000003EE 4D
213    000003EF 49
213    000003F0 4E
213    000003F1 49
213    000003F2 54
214    000003F4 1206         DC.W  GVR
215    000003F6 0000 0000    DC.L  BOOT_MINIT
216                          REFR  BOOT_MREAD
217    000003FA 0A           DC.B  10,'BOOT_MREAD'
217    000003FB 42
217    000003FC 4F
217    000003FD 4F
217    000003FE 54
217    000003FF 5F
217    00000400 4D
217    00000401 52
217    00000402 45
217    00000403 41
217    00000404 44
218    00000406 1206         DC.W  GVR
```



```
219    00000408 0000 0000    DC.L  BOOT_MREAD
220                          REFR  EVALGVR
221    0000040C 07           DC.B  7,'EVALGVR'
221    0000040D 45
221    0000040E 56
221    0000040F 41
221    00000410 4C
221    00000411 47
221    00000412 56
221    00000413 52
222    00000414 1206         DC.W  GVR
223    00000416 0000 0000    DC.L  EVALGVR
224                          REFR  FS_FWRITESTRINT
225    0000041A 0F           DC.B  15,'FS_FWRITESTRINT'
225    0000041B 46
225    0000041C 53
225    0000041D 5F
225    0000041E 46
225    0000041F 57
225    00000420 52
225    00000421 49
225    00000422 54
225    00000423 45
225    00000424 53
225    00000425 54
225    00000426 52
225    00000427 49
225    00000428 4E
225    00000429 54
226    0000042A 1206         DC.W  GVR
227    0000042C 0000 0000    DC.L  FS_FWRITESTRINT
228                          REFR  G_DOLLAR
229    00000430 08           DC.B  8,'G_DOLLAR'
229    00000431 47
229    00000432 5F
229    00000433 44
229    00000434 4F
229    00000435 4C
229    00000436 4C
229    00000437 41
229    00000438 52
230    0000043A 1206         DC.W  GVR
231    0000043C 0000 0000    DC.L  G_DOLLAR
232                          REFR  INITLOAD
233    00000440 08           DC.B  8,'INITLOAD'
233    00000441 49
233    00000442 4E
233    00000443 49
233    00000444 54
233    00000445 4C
233    00000446 4F
233    00000447 41
233    00000448 44
234    0000044A 1206         DC.W  GVR
235    0000044C 0000 0000    DC.L  INITLOAD
236                          REFR  INITLOAD_INITLOAD
237    00000450 11           DC.B  17,'INITLOAD_INITLOAD'
```

```
237    00000451 49
237    00000452 4E
237    00000453 49
237    00000454 54
237    00000455 4C
237    00000456 4F
237    00000457 41
237    00000458 44
237    00000459 5F
237    0000045A 49
237    0000045B 4E
237    0000045C 49
237    0000045D 54
237    0000045E 4C
237    0000045F 4F
237    00000460 41
237    00000461 44
238    00000462 1206          DC.W  GVR
239    00000464 0000 0000     DC.L  INITLOAD_INITLOAD
240                           REFR  INITLOAD_SYSPREFIX
241    00000468 12            DC.B  18,'INITLOAD_SYSPREFIX'
241    00000469 49
241    0000046A 4E
241    0000046B 49
241    0000046C 54
241    0000046D 4C
241    0000046E 4F
241    0000046F 41
241    00000470 44
241    00000471 5F
241    00000472 53
241    00000473 59
241    00000474 53
241    00000475 50
241    00000476 52
241    00000477 45
241    00000478 46
241    00000479 49
241    0000047A 58
242    0000047C 1206          DC.W  GVR
243    0000047E 0000 0000     DC.L  INITLOAD_SYSPREFIX
244                           REFR  INITLOAD__BASE
245    00000482 0E            DC.B  14,'INITLOAD__BASE'
245    00000483 49
245    00000484 4E
245    00000485 49
245    00000486 54
245    00000487 4C
245    00000488 4F
245    00000489 41
245    0000048A 44
245    0000048B 5F
245    0000048C 5F
245    0000048D 42
245    0000048E 41
245    0000048F 53
245    00000490 45
```



```
246    00000492 1206          DC.W  GVR
247    00000494 0000 0000     DC.L  INITLOAD__BASE
248                           REFR  LOADER
249    00000498 06            DC.B  6,'LOADER'
249    00000499 4C
249    0000049A 4F
249    0000049B 41
249    0000049C 44
249    0000049D 45
249    0000049E 52
250    000004A0 1206          DC.W  GVR
251    000004A2 0000 0000     DC.L  LOADER
252                           REFR  LOADER_CHECKREV
253    000004A6 0F            DC.B  15,'LOADER_CHECKREV'
253    000004A7 4C
253    000004A8 4F
253    000004A9 41
253    000004AA 44
253    000004AB 45
253    000004AC 52
253    000004AD 5F
253    000004AE 43
253    000004AF 48
253    000004B0 45
253    000004B1 43
253    000004B2 4B
253    000004B3 52
253    000004B4 45
253    000004B5 56
254    000004B6 1206          DC.W  GVR
255    000004B8 0000 0000     DC.L  LOADER_CHECKREV
256                           REFR  LOADER_CLOSEFILES
257    000004BC 11            DC.B  17,'LOADER_CLOSEFILES'
257    000004BD 4C
257    000004BE 4F
257    000004BF 41
257    000004C0 44
257    000004C1 45
257    000004C2 52
257    000004C3 5F
257    000004C4 43
257    000004C5 4C
257    000004C6 4F
257    000004C7 53
257    000004C8 45
257    000004C9 46
257    000004CA 49
257    000004CB 4C
257    000004CC 45
257    000004CD 53
258    000004CE 1206          DC.W  GVR
259    000004D0 0000 0000     DC.L  LOADER_CLOSEFILES
260                           REFR  LOADER_COUNTCODE
261    000004D4 10            DC.B  16,'LOADER_COUNTCODE'
261    000004D5 4C
261    000004D6 4F
261    000004D7 41
```

```
 261    000004D8 44
 261    000004D9 45
 261    000004DA 52
 261    000004DB 5F
 261    000004DC 43
 261    000004DD 4F
 261    000004DE 55
 261    000004DF 4E
 261    000004E0 54
 261    000004E1 43
 261    000004E2 4F
 261    000004E3 44
 261    000004E4 45
 262    000004E6 1206          DC.W  GVR
 263    000004E8 0000 0000     DC.L  LOADER_COUNTCODE
 264                           REFR  LOADER_GETBYTES
 265    000004EC 0F            DC.B  15,'LOADER_GETBYTES'
 265    000004ED 4C
 265    000004EE 4F
 265    000004EF 41
 265    000004F0 44
 265    000004F1 45
 265    000004F2 52
 265    000004F3 5F
 265    000004F4 47
 265    000004F5 45
 265    000004F6 54
 265    000004F7 42
 265    000004F8 59
 265    000004F9 54
 265    000004FA 45
 265    000004FB 53
 266    000004FC 1206          DC.W  GVR
 267    000004FE 0000 0000     DC.L  LOADER_GETBYTES
 268                           REFR  LOADER_INITLOADER
 269    00000502 11            DC.B  17,'LOADER_INITLOADER'
 269    00000503 4C
 269    00000504 4F
 269    00000505 41
 269    00000506 44
 269    00000507 45
 269    00000508 52
 269    00000509 5F
 269    0000050A 49
 269    0000050B 4E
 269    0000050C 49
 269    0000050D 54
 269    0000050E 4C
 269    0000050F 4F
 269    00000510 41
 269    00000511 44
 269    00000512 45
 269    00000513 52
 270    00000514 1206          DC.W  GVR
 271    00000516 0000 0000     DC.L  LOADER_INITLOADER
 272                           REFR  LOADER_LOADER
 273    0000051A 0D            DC.B  13,'LOADER_LOADER'
```



```
 273    0000051B 4C
 273    0000051C 4F
 273    0000051D 41
 273    0000051E 44
 273    0000051F 45
 273    00000520 52
 273    00000521 5F
 273    00000522 4C
 273    00000523 4F
 273    00000524 41
 273    00000525 44
 273    00000526 45
 273    00000527 52
 274    00000528 1206          DC.W  GVR
 275    0000052A 0000 0000     DC.L  LOADER_LOADER
 276                           REFR  LOADER_LOADINFO
 277    0000052E 0F            DC.B  15,'LOADER_LOADINFO'
 277    0000052F 4C
 277    00000530 4F
 277    00000531 41
 277    00000532 44
 277    00000533 45
 277    00000534 52
 277    00000535 5F
 277    00000536 4C
 277    00000537 4F
 277    00000538 41
 277    00000539 44
 277    0000053A 49
 277    0000053B 4E
 277    0000053C 46
 277    0000053D 4F
 278    0000053E 1206          DC.W  GVR
 279    00000540 0000 0000     DC.L  LOADER_LOADINFO
 280                           REFR  LOADER_LOADQ
 281    00000544 0C            DC.B  12,'LOADER_LOADQ'
 281    00000545 4C
 281    00000546 4F
 281    00000547 41
 281    00000548 44
 281    00000549 45
 281    0000054A 52
 281    0000054B 5F
 281    0000054C 4C
 281    0000054D 4F
 281    0000054E 41
 281    0000054F 44
 281    00000550 51
 282    00000552 1206          DC.W  GVR
 283    00000554 0000 0000     DC.L  LOADER_LOADQ
 284                           REFR  LOADER_LOADTEXT
 285    00000558 0F            DC.B  15,'LOADER_LOADTEXT'
 285    00000559 4C
 285    0000055A 4F
 285    0000055B 41
 285    0000055C 44
 285    0000055D 45
```

```
285    0000055E 52
285    0000055F 5F
285    00000560 4C
285    00000561 4F
285    00000562 41
285    00000563 44
285    00000564 54
285    00000565 45
285    00000566 58
285    00000567 54
286    00000568 1206          DC.W  GVR
287    0000056A 0000 0000     DC.L  LOADER_LOADTEXT
288                           REFR  LOADER_MARKUSER
289    0000056E 0F            DC.B  15,'LOADER_MARKUSER'
289    0000056F 4C
289    00000570 4F
289    00000571 41
289    00000572 44
289    00000573 45
289    00000574 52
289    00000575 5F
289    00000576 4D
289    00000577 41
289    00000578 52
289    00000579 4B
289    0000057A 55
289    0000057B 53
289    0000057C 45
289    0000057D 52
290    0000057E 1206          DC.W  GVR
291    00000580 0000 0000     DC.L  LOADER_MARKUSER
292                           REFR  LOADER_MATCHFILE
293    00000584 10            DC.B  16,'LOADER_MATCHFILE'
293    00000585 4C
293    00000586 4F
293    00000587 41
293    00000588 44
293    00000589 45
293    0000058A 52
293    0000058B 5F
293    0000058C 4D
293    0000058D 41
293    0000058E 54
293    0000058F 43
293    00000590 48
293    00000591 46
293    00000592 49
293    00000593 4C
293    00000594 45
294    00000596 1206          DC.W  GVR
295    00000598 0000 0000     DC.L  LOADER_MATCHFILE
296                           REFR  LOADER_MOVEDEFS
297    0000059C 0F            DC.B  15,'LOADER_MOVEDEFS'
297    0000059D 4C
297    0000059E 4F
297    0000059F 41
297    000005A0 44
```



```
297    000005A1 45
297    000005A2 52
297    000005A3 5F
297    000005A4 4D
297    000005A5 4F
297    000005A6 56
297    000005A7 45
297    000005A8 44
297    000005A9 45
297    000005AA 46
297    000005AB 53
298    000005AC 1206          DC.W  GVR
299    000005AE 0000 0000     DC.L  LOADER_MOVEDEFS
300                           REFR  LOADER_OPENLINKF
301    000005B2 10            DC.B  16,'LOADER_OPENLINKF'
301    000005B3 4C
301    000005B4 4F
301    000005B5 41
301    000005B6 44
301    000005B7 45
301    000005B8 52
301    000005B9 5F
301    000005BA 4F
301    000005BB 50
301    000005BC 45
301    000005BD 4E
301    000005BE 4C
301    000005BF 49
301    000005C0 4E
301    000005C1 4B
301    000005C2 46
302    000005C4 1206          DC.W  GVR
303    000005C6 0000 0000     DC.L  LOADER_OPENLINKF
304                           REFR  LOADER_RELEASEUSER
305    000005CA 12            DC.B  18,'LOADER_RELEASEUSER'
305    000005CB 4C
305    000005CC 4F
305    000005CD 41
305    000005CE 44
305    000005CF 45
305    000005D0 52
305    000005D1 5F
305    000005D2 52
305    000005D3 45
305    000005D4 4C
305    000005D5 45
305    000005D6 41
305    000005D7 53
305    000005D8 45
305    000005D9 55
305    000005DA 53
305    000005DB 45
305    000005DC 52
306    000005DE 1206          DC.W  GVR
307    000005E0 0000 0000     DC.L  LOADER_RELEASEUSER
308                           REFR  LOADER_ZEROMEM
309    000005E4 0E            DC.B  14,'LOADER_ZEROMEM'
```

```
 309    000005E5 4C
 309    000005E6 4F
 309    000005E7 41
 309    000005E8 44
 309    000005E9 45
 309    000005EA 52
 309    000005EB 5F
 309    000005EC 5A
 309    000005ED 45
 309    000005EE 52
 309    000005EF 4F
 309    000005F0 4D
 309    000005F1 45
 309    000005F2 4D
 310    000005F4 1206          DC.W  GVR
 311    000005F6 0000 0000     DC.L  LOADER_ZEROMEM
 312                           REFR  LOADER__BASE
 313    000005FA 0C            DC.B  12,'LOADER__BASE'
 313    000005FB 4C
 313    000005FC 4F
 313    000005FD 41
 313    000005FE 44
 313    000005FF 45
 313    00000600 52
 313    00000601 5F
 313    00000602 5F
 313    00000603 42
 313    00000604 41
 313    00000605 53
 313    00000606 45
 314    00000608 1206          DC.W  GVR
 315    0000060A 0000 0000     DC.L  LOADER__BASE
 316                           REFR  MATCHDEFEXT
 317    0000060E 0B            DC.B  11,'MATCHDEFEXT'
 317    0000060F 4D
 317    00000610 41
 317    00000611 54
 317    00000612 43
 317    00000613 48
 317    00000614 44
 317    00000615 45
 317    00000616 46
 317    00000617 45
 317    00000618 58
 317    00000619 54
 318    0000061A 1206          DC.W  GVR
 319    0000061C 0000 0000     DC.L  MATCHDEFEXT
 320                           REFR  MINI
 321    00000620 04            DC.B  4,'MINI'
 321    00000621 4D
 321    00000622 49
 321    00000623 4E
 321    00000624 49
 322    00000626 1206          DC.W  GVR
 323    00000628 0000 0000     DC.L  MINI
 324                           REFR  MINI_IORESC
 325    0000062C 0B            DC.B  11,'MINI_IORESC'
```



```
 325    0000062D 4D
 325    0000062E 49
 325    0000062F 4E
 325    00000630 49
 325    00000631 5F
 325    00000632 49
 325    00000633 4F
 325    00000634 52
 325    00000635 45
 325    00000636 53
 325    00000637 43
 326    00000638 1206          DC.W  GVR
 327    0000063A 0000 0000     DC.L  MINI_IORESC
 328                           REFR  MINI_IORVAL
 329    0000063E 0B            DC.B  11,'MINI_IORVAL'
 329    0000063F 4D
 329    00000640 49
 329    00000641 4E
 329    00000642 49
 329    00000643 5F
 329    00000644 49
 329    00000645 4F
 329    00000646 52
 329    00000647 56
 329    00000648 41
 329    00000649 4C
 330    0000064A 1206          DC.W  GVR
 331    0000064C 0000 0000     DC.L  MINI_IORVAL
 332                           REFR  MINI_MINI
 333    00000650 09            DC.B  9,'MINI_MINI'
 333    00000651 4D
 333    00000652 49
 333    00000653 4E
 333    00000654 49
 333    00000655 5F
 333    00000656 4D
 333    00000657 49
 333    00000658 4E
 333    00000659 49
 334    0000065A 1206          DC.W  GVR
 335    0000065C 0000 0000     DC.L  MINI_MINI
 336                           REFR  MINI_MINIIO
 337    00000660 0B            DC.B  11,'MINI_MINIIO'
 337    00000661 4D
 337    00000662 49
 337    00000663 4E
 337    00000664 49
 337    00000665 5F
 337    00000666 4D
 337    00000667 49
 337    00000668 4E
 337    00000669 49
 337    0000066A 49
 337    0000066B 4F
 338    0000066C 1206          DC.W  GVR
 339    0000066E 0000 0000     DC.L  MINI_MINIIO
 340                           REFR  MINI__BASE
```

```
341    00000672 0A           DC.B  10,'MINI__BASE'
341    00000673 4D
341    00000674 49
341    00000675 4E
341    00000676 49
341    00000677 5F
341    00000678 5F
341    00000679 42
341    0000067A 41
341    0000067B 53
341    0000067C 45
342    0000067E 1206         DC.W  GVR
343    00000680 0000 0000    DC.L  MINI__BASE
344                          REFR  RELOCATE
345    00000684 08           DC.B  8,'RELOCATE'
345    00000685 52
345    00000686 45
345    00000687 4C
345    00000688 4F
345    00000689 43
345    0000068A 41
345    0000068B 54
345    0000068C 45
346    0000068E 1206         DC.W  GVR
347    00000690 0000 0000    DC.L  RELOCATE
348                          REFR  STACKFUDGE
349    00000694 0A           DC.B  10,'STACKFUDGE'
349    00000695 53
349    00000696 54
349    00000697 41
349    00000698 43
349    00000699 4B
349    0000069A 46
349    0000069B 55
349    0000069C 44
349    0000069D 47
349    0000069E 45
350    000006A0 1206         DC.W  GVR
351    000006A2 0000 0000    DC.L  STACKFUDGE
352                          REFR  SYSGLOBALS
353    000006A6 0A           DC.B  10,'SYSGLOBALS'
353    000006A7 53
353    000006A8 59
353    000006A9 53
353    000006AA 47
353    000006AB 4C
353    000006AC 4F
353    000006AD 42
353    000006AE 41
353    000006AF 4C
353    000006B0 53
354    000006B2 1206         DC.W  GVR
355    000006B4 0000 0000    DC.L  SYSGLOBALS
356                          REFR  SYSGLOBALS_FSIDC
357    000006B8 10           DC.B  16,'SYSGLOBALS_FSIDC'
357    000006B9 53
357    000006BA 59
```



```
357    000006BB 53
357    000006BC 47
357    000006BD 4C
357    000006BE 4F
357    000006BF 42
357    000006C0 41
357    000006C1 4C
357    000006C2 53
357    000006C3 5F
357    000006C4 46
357    000006C5 53
357    000006C6 49
357    000006C7 44
357    000006C8 43
358    000006CA 1206         DC.W  GVR
359    000006CC 0000 0000    DC.L  SYSGLOBALS_FSIDC
360                          REFR  SYSGLOBALS_SYSGLOBALS
361    000006D0 15           DC.B  21,'SYSGLOBALS_SYSGLOBALS'
361    000006D1 53
361    000006D2 59
361    000006D3 53
361    000006D4 47
361    000006D5 4C
361    000006D6 4F
361    000006D7 42
361    000006D8 41
361    000006D9 4C
361    000006DA 53
361    000006DB 5F
361    000006DC 53
361    000006DD 59
361    000006DE 53
361    000006DF 47
361    000006E0 4C
361    000006E1 4F
361    000006E2 42
361    000006E3 41
361    000006E4 4C
361    000006E5 53
362    000006E6 1206         DC.W  GVR
363    000006E8 0000 0000    DC.L  SYSGLOBALS_SYSGLOBALS
364                          REFR  SYSGLOBALS__BASE
365    000006EC 10           DC.B  16,'SYSGLOBALS__BASE'
365    000006ED 53
365    000006EE 59
365    000006EF 53
365    000006F0 47
365    000006F1 4C
365    000006F2 4F
365    000006F3 42
365    000006F4 41
365    000006F5 4C
365    000006F6 53
365    000006F7 5F
365    000006F8 5F
365    000006F9 42
365    000006FA 41
```

```
  365    000006FB 53
  365    000006FC 45
  366    000006FE 1206          DC.W  GVR
  367    00000700 0000 0000     DC.L  SYSGLOBALS__BASE
  368
  369              0000 0704 endd    equ *
  370
  371                                end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ALPHALIST	REL	20	00000002
ASM_ASM	REL	24	00000005
ASM_ASSIGN	REL	28	00000007
ASM_CI_SWITCH	REL	32	0000000A
ASM_CPYMSG	REL	36	0000000E
ASM_DIFFERENCE	REL	40	00000011
ASM_DIV	REL	44	00000015
ASM_EQUAL	REL	48	00000017
ASM_ERRMSG	REL	52	0000001A
ASM_FASTMOVE	REL	56	0000001D
ASM_FINDROMS	REL	60	00000021
ASM_FLPYINIT	REL	64	00000025
ASM_FLPYMREAD	REL	68	00000029
ASM_FLPYMWRITE	REL	72	0000002D
ASM_FLPYREAD	REL	76	00000031
ASM_FLPY_WRT	REL	80	00000035
ASM_F_PWR_ON	REL	84	00000039
ASM_IN	REL	88	0000003D
ASM_INCLUSION	REL	92	0000003F
ASM_INTERSECT	REL	96	00000043
ASM_MEMAVAIL	REL	100	00000047
ASM_MOD	REL	104	0000004B
ASM_MOVEL	REL	108	0000004D
ASM_MOVELEFT	REL	112	00000050
ASM_MOVER	REL	116	00000054
ASM_MOVERIGHT	REL	120	00000057
ASM_MPY	REL	124	0000005B
ASM_NEQUAL	REL	128	0000005D
ASM_NEWBYTES	REL	132	00000060
ASM_NEWWORDS	REL	136	00000064
ASM_POS	REL	140	00000068
ASM_POWERUP	REL	144	0000006A
ASM_RMOVEL	REL	148	0000006D
ASM_RMOVER	REL	152	00000070
ASM_UNION	REL	156	00000073
ASM_USERPROGRAM	REL	160	00000076
BOOTDAMMODULE	REL	164	0000007A
BOOTDAMMODULE_BOOTDAM	REL	168	0000007E
BOOTDAMMODULE_BOOTDAMMODULE	REL	172	00000084
BOOTDAMMODULE_BOOTNODE	REL	176	0000008B
BOOTDAMMODULE_BOOTTM	REL	180	00000091
BOOTDAMMODULE_INITBOOTDAM	REL	184	00000097
BOOTDAMMODULE_SRMNODE	REL	188	0000009E
BOOTDAMMODULE__BASE	REL	192	000000A4
BOOT_FINDFILE	REL	196	000000A9
BOOT_LIFHEAD	REL	200	000000AD
BOOT_MFCLOSE	REL	204	000000B1
BOOT_MFOPEN	REL	208	000000B5
BOOT_MINIT	REL	212	000000B8
BOOT_MREAD	REL	216	000000BB
EVALGVR	REL	220	000000BE

```
FS_FWRITESTRINT REL      224    000000C0
G_DOLLAR        REL      228    000000C4
INITLOAD        REL      232    000000C7
INITLOAD_INITLOAD REL      236    000000CA
INITLOAD_SYSPREFIX REL      240    000000CF
INITLOAD__BASE  REL      244    000000D4
LOADER          REL      248    000000D8
LOADER_CHECKREV REL      252    000000DA
LOADER_CLOSEFILES REL      256    000000DE
LOADER_COUNTCODE REL      260    000000E3
LOADER_GETBYTES REL      264    000000E8
LOADER_INITLOADER REL      268    000000EC
LOADER_LOADER   REL      272    000000F1
LOADER_LOADINFO REL      276    000000F5
LOADER_LOADQ    REL      280    000000F9
LOADER_LOADTEXT REL      284    000000FD
LOADER_MARKUSER REL      288    00000101
LOADER_MATCHFILE REL      292    00000105
LOADER_MOVEDEFS REL      296    0000010A
LOADER_OPENLINKF REL      300    0000010E
LOADER_RELEASEUSER REL      304    00000113
LOADER_ZEROMEM  REL      308    00000118
LOADER__BASE    REL      312    0000011C
MATCHDEFEXT     REL      316    00000120
MINI            REL      320    00000123
MINI_IORESC     REL      324    00000125
MINI_IORVAL     REL      328    00000128
MINI_MINI       REL      332    0000012B
MINI_MINIIO     REL      336    0000012E
MINI__BASE      REL      340    00000131
RELOCATE        REL      344    00000134
STACKFUDGE      REL      348    00000137
SYSGLOBALS      REL      352    0000013A
SYSGLOBALS_FSIDC REL      356    0000013D
SYSGLOBALS_SYSGLOBALS REL      360    00000142
SYSGLOBALS__BASE REL      364    00000148
```

INTERNAL SYMBOLS

SYMBOL	TYPE	DEF	EQU SYM	VALUE
A0	AREG	0		00000000
A1	AREG	0		00000001
A2	AREG	0		00000002
A3	AREG	0		00000003
A4	AREG	0		00000004
A5	AREG	0		00000005
A6	AREG	0		00000006
A7	AREG	0		00000007
CCR	STREG	0		00000005
D0	DREG	0		00000000
D1	DREG	0		00000001
D2	DREG	0		00000002
D3	DREG	0		00000003
D4	DREG	0		00000004
D5	DREG	0		00000005
D6	DREG	0		00000006
D7	DREG	0		00000007


```
DEFADDR         REL       17                     00000028
ENDD            REL      369                     00000704
GVR             ABS       15                     00001206
SP              AREG       0                     00000007
SR              STREG      0                     00000006
SYSDEFTABLE     REL        5                     00000000
USP             STREG      0                     00000007
X               REL       13                     00000025
```

Part No. 09826-90074
E0383
Microfiche No. 09826-99074

Printed in U.S.A.
First Edition, March 1983