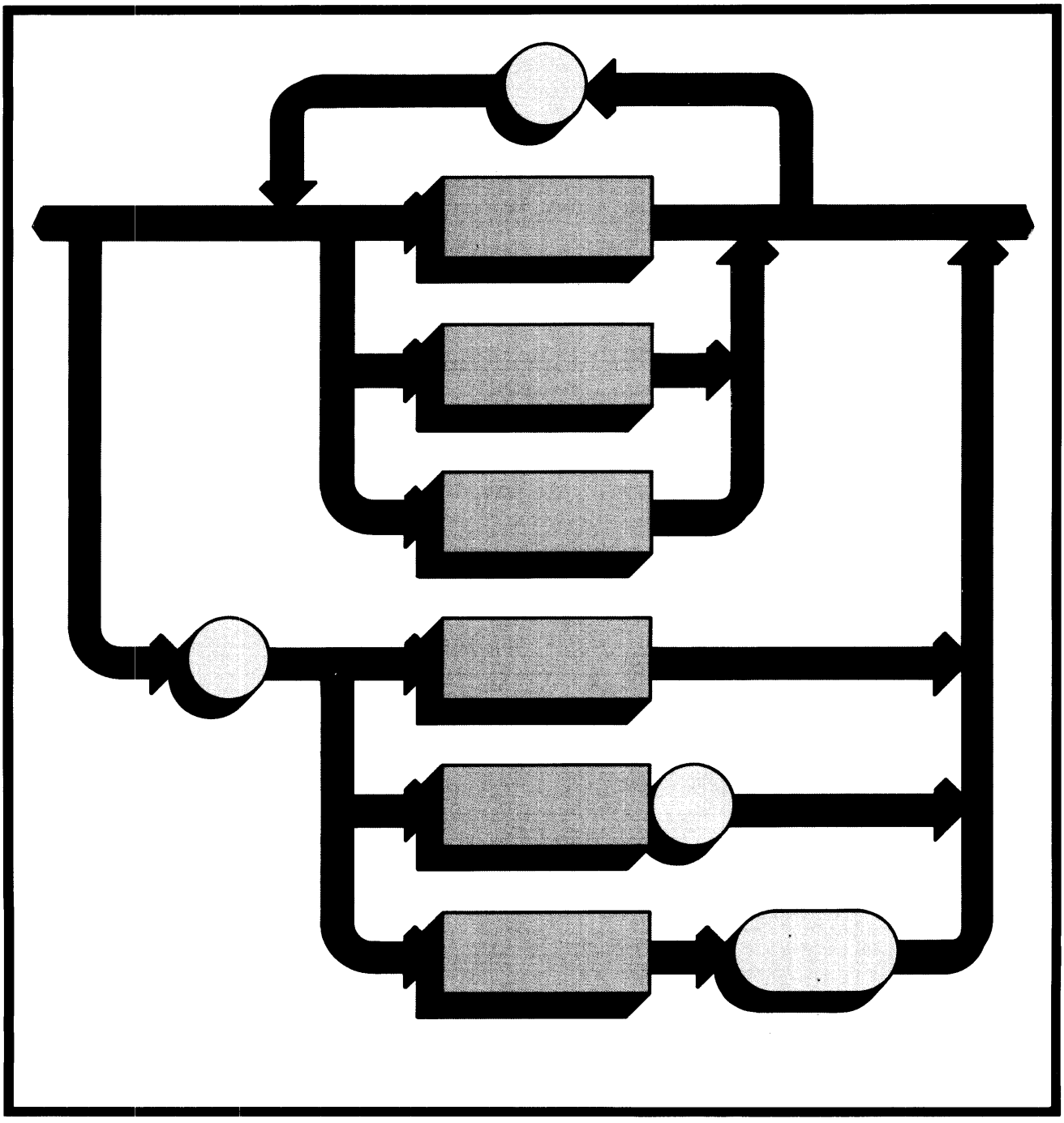
HP 9000 Series 200 Computers

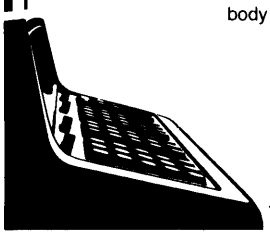
HEWLETT PACKARD

Pascal Source Code Listings
Volume I-Assembly Language



Pascal 3.0 Source Code Listings
Volume I-Assembly
for the HP 9000 Series 200 Computers

Manual Part No. 98615-90074



Hewlett-Packard Company
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.

February 1985...Edition 1

Table of Contents

```
asm_cache_on
  POWERUP              15    18   164
  ROMCALL               4    93   216
  SYSDEF               36    39
asm_ci_switch
  ASM                   8   204
  SYSDEF               40    43
asm_closefiles
  POWERUP               9   301   317   344   589
asm_copymsg
  ASM                   8   260
  SYSDEF               44    47
asm_delete
  STRG2               142   143
asm_difference
  BOOTDEFS             52    60
  SYSDEF               48    51
asm_div
  MODIV                 2    22
  SYSDEF               52    55
asm_equal
  BOOTDEFS             53    61
  SYSDEF               56    59
asm_errmsg
  ASM                   7   291
  SYSDEF               60    63
asm_f_pwr_on
  ROMCALL               3   120
  SYSDEF               92    95
asm_fastmove
  FASTMOVE              -    32
  SYSDEF               64    67
asm_findroms
  ASM                   8   214
  SYSDEF               68    71
asm_flpy_wrt
  ROMCALL               3   132
  SYSDEF               88    91
asm_flpyinit
  ROMCALL               3    63
  SYSDEF               72    75
asm_flpymread
  ROMCALL               4   152
  SYSDEF               76    79
asm_flpymwrite
  ROMCALL               4   148
  SYSDEF               80    83
asm_flpyread
  ROMCALL               3   136   183
  SYSDEF               84    87
asm_iand
  POWERUP              12   676
asm_ior
  BOOTDEFS             53    62
  SYSDEF               96    99
asm_inclusion
  BOOTDEFS             53    63
  SYSDEF              100   103
asm_init
  RS                  541  1086
asm_initvects
  POWERUP               9   129
asm_insert
  STRG2                55    56
```

```
asm_intersect
  BOOTDEFS             54    64
  SYSDEF              104   107
asm_intlevel
  POWERUP               8   655
asm_ior
  POWERUP              12   682
asm_memavail
  NEWWORDS              6    12
  SYSDEF              108   111
asm_mod
  MODIV                 3    17
  SYSDEF              112   115
asm_movel
  FASTMOVE              3    14
  SYSDEF              116   119
asm_moveleft
  FASTMOVE              3    15
  SYSDEF              120   123
asm_mover
  FASTMOVE              3    23
  SYSDEF              124   127
asm_moveright
  FASTMOVE              3    24
  SYSDEF              128   131
asm_mpy
  BOOTDEFS             54    65
  SYSDEF              132   135
asm_nequal
  BOOTDEFS             54    66
  SYSDEF              136   139
asm_newbytes
  NEWWORDS              6    28
  SYSDEF              140   143
asm_newwords
  NEWWORDS              6    22
  SYSDEF              144   147
asm_pos
  BOOTDEFS             57    71
  SYSDEF              148   151
asm_powerup
  ASM                   7    98
  SYSDEF              152   155
asm_psubtopsub
  STRG1                29    30
asm_psubtossub
  STRG1               195   196
asm_rmovel
  BOOTDEFS             56    69
  SYSDEF              156   159
asm_rmover
  BOOTDEFS             56    70
  SYSDEF              160   163
asm_sappend
  STRG2                25    26
asm_scan
  SCAN                 17    18
asm_scopy
  STRG2               105   106
asm_setassign
  SETSTUFF              2     4
asm_setintlevel
  POWERUP               8   664
```

```
di_cpfi
  DISCINT          *  1253
di_tcfo
  DISCINT             1247  1258
di_cir
  DISCINT              524  1057
di_dma_en
  DISCINT             1434  1449
di_dma_w
  DISCINT          *  1451
di_dma_w1
  DISCINT             1454  1455
di_dma_w2
  DISCINT             1452  1457
di_dmyrts
  DISCINT             1064  1066  1079
di_epp
  DISCINT             1005  1043
di_fcc
  DISCINT          *  1282
di_frw
  DISCINT             1152  1274
di_frwc
  DISCINT             1277  1287
di_ifc
  DISCINT              609  1003  1024
di_init
  DISCINT              387   607
di_init_s
  DISCINT              608   625   926
di_isr
  DISCINT              396  1147
di_isr1
  DISCINT             1148  1299
di_isr_ex
  DISCINT             1158  1160
di_isrdma
  DISCINT             1156  1166
di_local
  DISCINT             1063  1076
di_log
  DISCINT             1306  1314  1322  1330  1338
di_loop
  GPIODVR              307   310
di_lsterr
  DISCINT              663   728
di_no_del
  DISCINT             1305  1308
di_no_pol
  DISCINT             1321  1324
di_no_stg
  DISCINT             1329  1332
di_no_stg
  DISCINT             1313  1316
di_opdma
  DISCINT             1365  1367
di_optautl
  DISCINT              713   970  1123
di_optsutl
  DISCINT              716  1016  1025  1077
di_noword
  DISCINT              725  1357
di_p_poll
  DISCINT              502   969

di_rbout
  DISCINT              492  1120
di_rdb
  DISCINT              406   429   431   661
di_rdbl
  DISCINT              665   668
di_ras
  DISCINT              458   826
di_ren
  DISCINT             1002  1015
di_reni
  GPIODVR              296   300
di_rqs
  DISCINT              913   936
di_rqs2
  DISCINT              939   942
di_sc_err
  DISCINT              710   997  1004  1006  1007  1008  1009  1058  1065  1067  1068  1069  1070  1102
di_scbsy
  DISCINT          *   707
di_set
  DISCINT              514   996
di_t_bst
  DISCINT             1382  1383  1394  1470
di_t_dma
  DISCINT             1387  1389  1393  1395  1396  1401
di_t_fhs
  DISCINT             1376  1377  1388  1477
di_t_int
  DISCINT             1380  1384  1392  1463
di_test
  DISCINT              536  1102
di_tfr
  DISCINT              482  1354
di_ti_term
  DISCINT          *  1263
di_tid
  DISCINT          *  1430
di_tlkerr
  DISCINT              690   731
di_tmo
  DISCINT              734   788
di_to_term
  DISCINT             1265  1288
di_tod
  DISCINT             1425  1440
di_tod_l
  DISCINT             1444  1446  1448
di_wait_fb
  DISCINT              668   754
di_wait_fi
  DISCINT              691   751   972  1125
di_wfc
  DISCINT              752   759
di_wfc_done
  DISCINT              770   782   797   814
di_wfc_infinite
  DISCINT              776   792   793
di_wfc_quick
  DISCINT              769   771
di_wfc_exit
  DISCINT              806   813
di_wfc_timed
  DISCINT              781   784   812
```

```
extdc_direct_status
  DC                    176   234
extdc_enter_data
  DC                    173   231
extdc_extdc
  DC                    169   224
extdc_output_data
  DC                    174   232
extdc_output_enc
  DC                    175   233
extdc_start_tfr_ir
  DC                    179   237
extdc_start_tfr_out
  DC                    180   238
extdi
  DISCINT           *    77
extdi_edi_clr
  DISCINT               123   519
extdi_edi_end
  DISCINT               125   543
extdi_edi_init
  DISCINT               110   383
extdi_edi_isr
  DISCINT               111   392
extdi_edi_ppoll
  DISCINT               121   497
extdi_edi_rdb
  DISCINT               112   401
extdi_edi_rds
  DISCINT               116   452
extdi_edi_rdw
  DISCINT               114   424
extdi_edi_send
  DISCINT               120   487
extdi_edi_set
  DISCINT               122   509
extdi_edi_test
  DISCINT               124   530
extdi_edi_tfr
  DISCINT               118   477
extdi_edi_wtb
  DISCINT               113   413
extdi_edi_wtc
  DISCINT               117   465
extdi_edi_wtw
  DISCINT               115   438
extdi_extdi
  DISCINT               109   378
extend
  STRINT                 12    75    82
extg
  GPIO              *   104
extg_eg_clr
  GPIO                  139   486
extg_eg_init
  GPIO                  132   371
extg_eg_isr
  GPIO                  133   380
extg_eg_rdb
  GPIO                  134   398
extg_eg_rds
  GPIO                  136   442
extg_eg_rdw
  GPIO                  135   420
```

```
extg_eg_set
  GPIO                  139   476
extg_eg_tdma
  GPIO                  133   389   954
extg_eg_test
  GPIO                  139   496
extg_eg_tfr
  GPIO                  137   466
extg_eg_wtb
  GPIO                  134   410
extg_eg_wtc
  GPIO                  136   455
extg_eg_wtw
  GPIO                  135   432
extg_extg
  GPIO                  130   366
exth
  HPIB              *   148
exth_eh_clr
  HPIB                  187   582
exth_eh_end
  HPIB                  187   605
exth_eh_init
  HPIB                  179   437
exth_eh_isr
  HPIB                  180   446
exth_eh_ppoll
  HPIB                  186   560
exth_eh_rdb
  HPIB                  181   464
exth_eh_rds
  HPIB                  183   516
exth_eh_rdw
  HPIB                  182   486
exth_eh_send
  HPIB                  186   550
exth_eh_set
  HPIB                  186   572
exth_eh_tdma
  HPIB                  180   455  2488
exth_eh_test
  HPIB                  187   592
exth_eh_tfr
  HPIB                  184   540
exth_eh_wtb
  HPIB                  181   476
exth_eh_wtc
  HPIB                  183   529
exth_eh_wtw
  HPIB                  182   502
exth_exth
  HPIB                  177   432
exxiit
  SETSTUFF              105   111
f
  ROMCALL                32    58    60
f2_16
  ROMCALL           *    38
f2_4
  ROMCALL           *    50
f85_grs
  FMINIT                648   690
f85_rb_i
  FMINIT                668   674
```

```
h_bustat
  HPIB                639  1111  1561  2049
h_bytist
  HPIB          *    1707
h_c_tbl
  HPIB               1493  1498  1499  1500  1501  1502  1503  1504  1505  1506
h_chka1
  HPIB               2025  2028
h_chka2
  HPIB               2029  2032
h_chkaddr
  HPIB               1274  1933  2024
h_chks1
  HPIB          *    2053
h_chks2
  HPIB               2048  2050  2052  2054
h_chksrq
  HPIB               1275  1928  2047
h_clr
  HPIB                588  1490
h_clr_atn
  HPIB               1506  1535
h_cmdpass
  HPIB                643  1125  1352  1945
h_crdreg
  HPIB               1051  1118  1125
h_dacr0
  HPIB                659  1932  1937  1946  1982
h_dacr1
  HPIB          *    660
h_das0
  HPIB                687  1281  1312  2184  2466
h_das1
  HPIB                688  1256  1303  1326  2132  2295
h_datain
  HPIB                645  1113  1742  1813  2156  2293  2368
h_dataout
  HPIB                646   821   839  1772  1860
h_disable
  HPIB               1325  1785
h_dma_w
  HPIB               2185  2201  2235  2253
h_dma_w1
  HPIB               2204  2205
h_dma_w2
  HPIB               2203  2206
h_dmaterm
  HPIB                460  1631
h_dmati_1
  HPIB               1649  1657
h_dmato
  HPIB               1644  1653
h_dmato_1
  HPIB               1655  1658
h_dmxrts
  HPIB               1479  1500  1502
h_ed_com
  HPIB               1305  1329
h_ed_exit
  HPIB               1310  1312
h_eir
  HPIB                796  1149  1255
h_eir2
  HPIB               1266  1268
```

```
h_eir3
  HPIB               1272  1274
h_eir4
  HPIB               1279  1281
h_eir5
  HPIB               1283  1285
h_enable
  HPIB               1302  1650  2233  2252
h_eoi
  HPIB               1379  1478
h_eot1
  COMASM              577   606
h_eot2
  COMASM              583   585
h_eot3
  COMASM              578   589
h_extstat
  HPIB                633   754   759   798  1099  1179  1396  1418  1450  1519  1535  1614  2047  2133
h_fakeisr
  HPIB                905   960  1011  1029  1280  1311  2364
h_feoi
  HPIB                671  1478  1771  1859  2440
h_fget0
  HPIB          *    667
h_fget1
  HPIB          *    668
h_flags
  HPIB                407   815   933  1220  1225  1231  1270  1282  1456  1582  1606  1676  1687  1706  1781  1923  1938  1983
                     1998  2060  2143  2161  2240  2464
h_fri
  HPIB          *    1800
h_frw
  HPIB               1682  1792
h_frw_other
  HPIB               1810  1817  1831  1852
h_frwi_1
  HPIB               1804  1805  1811  1829
h_frwi_2
  HPIB               1807  1812
h_frwi_2a
  HPIB               1803  1809  1813
h_frwi_3
  HPIB               1816  1819  1821  1824
h_frwo
  HPIB               1795  1846
h_frwo_1
  HPIB               1848  1849  1854  1862
h_frwo_2
  HPIB               1847  1851  1855
h_frwo_3
  HPIB               1856  1858  1860
h_g_stat1
  HPIB               1010  1013
h_getstat
  HPIB                918   925   940   973   980   986  1008
h_gts
  HPIB                676   812   837  1184  1354  1426  1537  1612  1966  2077
h_hdfa0
  HPIB                662  2177  2311
h_hdfa1
  HPIB                663   788  1647  1704  2340
h_hdfe0
  HPIB                664   789  1646  1705
```

```
h_hdfe1
  HPIB                665  2176
h_ifc
  HPIB                766  1377  1418
h_ifc2
  HPIB               1398  1427
h_init
  HPIB                442   751
h_init0
  HPIB                758   762   764
h_init_c
  HPIB                755   757
h_init_s
  HPIB                765   777  1180
h_int0copy
  HPIB                401   786   907   929   930   936   962  1013  1014  1032  1121  1276  1307  1345  1462  1708  1907  1913
                     2338  2349  2380  2436  2452  2465
h_int0mask
  HPIB                635   785  1269  1306
h_int0stat
  HPIB                634   902   957  1008  1677  1804  1812  1848  2291  2366
h_int1copy
  HPIB           *    402
h_int1mask
  HPIB           *    637
h_int1stat
  HPIB                636  1906
h_int_ca
  COMASM              694   814
  HPIB                622   757
h_intmsksav
  HPIB                403  1122  1268  1277  1304  1305  1308  1327  1909  1910
h_isr
  HPIB                451  1034  1675
h_isr0
  HPIB               1833  1905
h_isr1
  HPIB               1678  1907
h_isr_ex
  HPIB               1688  1690
h_isr_pm
  COMASM         *    234
  DC             *    621
  DISCINT        *    224
  GPIO           *    242
  HPIB           *    284
  RS             *    182
h_isr_pr
  COMASM              232   605
  DC             *    619
  DISCINT        *    222
  GPIO           *    240
  HPIB           *    282
  RS             *    180
h_isr_sl
  COMASM         *    233
  DC             *    620
  DISCINT        *    223
  GPIO           *    241
  HPIB           *    283
  RS             *    181
h_isrdma
  HPIB               1686  1695
```

```
h_local
  HPIB               1499  1519
h_log
  HPIB               1924  1941  1969  1984  1994  2027  2031  2053  2060
h_lon0
  HPIB                672  1596
h_lon1
  HPIB                673  1588
h_lsterr
  HPIB                814   863
h_nbaf
  HPIB           *    666
h_no_dcl
  HPIB               1936  1940  1943
h_no_end
  HPIB               2002  2005
h_no_get
  HPIB               1981  1985
h_no_ifc
  HPIB               1922  1926
h_no_int0
  HPIB               1990  2006
h_no_int1
  HPIB               1919  1989
h_no_ma
  HPIB               1931  1935
h_no_ppc
  HPIB               1971  1973
h_no_ppe
  HPIB               1976  1978
h_no_rlc
  HPIB               1993  1996
h_no_spas
  HPIB               1997  2001
h_no_srq
  HPIB               1927  1930
h_no_tct
  HPIB               1949  1968  1970
h_no_ucg
  HPIB               1944  1980
h_nobyte
  HPIB               1709  1720
h_nodma
  HPIB               2085  2087
h_notactl
  HPIB                853  1451  1536
h_notsctl
  HPIB                855  1397  1419  1520
h_noword
  HPIB                861  2079
h_p_poll
  HPIB                566  1343
h_ppe
  HPIB               1974  1977
h_ppoll
  HPIB                644   792  1234  1999
h_ppollmsk
  HPIB                418   793  1189  1197  1198  1227  1999
h_pts
  HPIB                589  1972
h_r6out
  HPIB                556  1580
h_r6out2
  HPIB           *   1601
```

```
noc:4
  DC                1696  1709
npibaddr
  FORMAT              38   582
  VERIFY              32   436
hpi_wtc
  HPIB          *   1171
hpi_wtc0
  HPIB          *   1173
hpi_wtc1
  DISCINT       *    925
  HPIB              1173  1179
hpi_wtc2
  DISCINT             928   930
  HPIB               1182  1185
hpi_wtc3
  HPIB               1172  1187  1977
hpi_wtc4
  DISCINT       *    944
  HPIB               1190  1199  1228
hpi_wtc5
  DISCINT             948   950
  HPIB               1232  1234
hr_card
  HPIB               1062  1086
hr_card1
  HPIB          *   1087
hr_temp
  HPIB               1061  1079
htctloop
  HPIB               1614  1615
hterr_b
  DISCINT             719  1374  1378  1386  1390  1464  1471  1478
  HPIB                857  2094  2098  2106  2110
hterr_d
  DISCINT             722  1375  1381
  HPIB                859  2095  2101
hwtcttbl
  DISCINT             909   912
  HPIB               1142  1144
i
  ALPHALIST            15    38    42    44
  ASM                 277   282   285
  RSTRINT              17    23    91
i_loop
  ALPHALIST            42    74
  POWERUP              78   181
ibadformat
  RSTRINT               5    36
icmod
  FORMAT               66   253   264   273   282   285   296   305   314   317   328   362   376   384   394   398   424   444
                      458   482   493   585
  VERIFY               60   182   193   202   211   214   225   234   243   246   257   265   276   316   330   356   367   438
icount
  ALPHALIST            22    41    43    45
id
  FORMAT               46   581
  VERIFY               40   435
id_reg
  RS                  445   586   814   929  1174
idle
  BUS_DVR       *    638
  FORMAT              267   284   299   316   331   354   373   378   396   427   455   495
  VERIFY              196   213   228   245   260   278   308   327   332   369
```

```
idle1
  BUS_DVR             640   641
ifclear
  FORMAT              583   592
  VERIFY              437   446
iflag
  FORMAT              128   403   410   417   430   515
  VERIFY        *    114
ignbus1
  POWERUP             295   297
ignore_be
  RS                  489   491   569  1019  1231  1394
ignorebus
  POWERUP             145   293   695   743
iiwait
  FORMAT              523   524
  VERIFY              379   380
iiwait2
  FORMAT              529   532   550
  VERIFY              385   388
imsk
  FORMAT               34   252   259   272   292   304   324   339   348   354   356   466   564
  VERIFY               28   181   188   201   221   233   253   282   293   302   308   310   339   418
in_0
  DC                 2518  2550  2581
in_1
  DC                 2512  2516  2522  2560
in_1b
  DC                 2525  2534
in_1c
  DC                 2532  2541
in_2
  DC                 2558  2562
in_3
  DC                 2554  2577
in_4
  DC                 2527  2530  2535  2538  2540  2547
in_ex2
  DC                 2565  2567
in_exit
  DC                 2520  2564  2578
in_isr
  RS                  475   476   563  1268  1501  2019
in_timeout
  DC                 2396  2506  2536
index
  RSTRINT               9    30    31    33    34    39    55    88    90
  STRG1         *     13
  STRG2                13    59    73    84   115   123   130   150   161   165
  STRINT               23    99   107   126   144
index_color
  ASM_SCLIP     *    144
  ASM_STEXT     *    133
  DGL_AUTL      *    154
inemp
  FORMAT        *    465
  VERIFY        *    338
info1
  ASM_SCLIP     *    145
  ASM_STEXT           134   231
  DGL_AUTL      *    155
```

```
llo
  DISCINT          *    588
  HPIB             *    709
lloop1
  FASTMOVE              65    69
lloop2
  FASTMOVE              73    75
lloop3
  FASTMOVE              87    88    90
loader
  ASM                   46    53
  EVALGVR                5     8
  POWERUP               15    33
  SYSDEF               256   259
loader_base
  SYSDEF               320   323
loader_checkrev
  SYSDEF               260   263
loader_closefiles
  SYSDEF               264   267
loader_countcode
  SYSDEF               268   271
loader_getbytes
  SYSDEF               272   275
loader_initloader
  SYSDEF               276   279
loader_loader
  SYSDEF               280   283
loader_loadinfo
  SYSDEF               284   287
loader_loadq
  SYSDEF               288   291
loader_loadtext
  SYSDEF               292   295
loader_markuser
  SYSDEF               296   299
loader_matchfile
  SYSDEF               300   303
loader_movedefs
  SYSDEF               304   307
loader_openlinkf
  SYSDEF               308   311
loader_releaseuser
  SYSDEF               312   315
loader_zeromem
  SYSDEF               316   319
loc_init
  DGL_AUTL         *     76
  DGL_TBODY             80    90
locals
  FMINIT                88   251
  FORMAT               134   137
  VERIFY               121   123
logeot
  COMASM                88   575
  DISCINT              141   153
  HPIB                 201   214
  RS                   112   113  1354  1360  1670
logint
  COMASM                87   605
  DISCINT              140   153  1159
  GPIO                 150   166  1118
  HPIB                 200   214  1273  1284  1689
```

```
longer
  STRINT               132   134
longoffset
  EVALGVR               19    81
look_for_timer
  POWERUP              739   776
loop1
  SETSTUFF              17    21
loop2
  SETSTUFF              31    34
loop3
  SETSTUFF              50    52
loop_last
  RS                  1345  1350
loop_thre
  RS                  2008  2012
loopt
  FORMAT               415   428
loopt2
  FORMAT               430   434
loopt3
  FORMAT               431   438
loopw
  FORMAT               410   413
loopw2
  FORMAT               417   420   437
lowcode
  ASM                   89   100
lowmem
  ASM                   67   101   110
  POWERUP          *    44
lp1
  POWERUP              634   635
ltrim0
  STRG1                287   291
ltrim1
  STRG1                285   290
ltrim2
  STRG1                284   288   293
ltrim3
  STRG1                297   299
lunched
  DC                   885  1189  1358  1456
m
  ASM                  263   333
m1
  ASM                  295   334
m68ktype
  ASM                    9    64    65   149   326
  DC                   199   924
  POWERUP              119   120   294   323   349   506   560
  STRINT                 2    73
  SYSDEF               324   327
m_bottom
  MODIV                124   134
m_top
  MODIV                120   124
ma
  COMASM           *   244
  DC               *   631
  DISCINT              234   872   923
  GPIO             *   252
  HPIB                 294  1102  1176  1585
  RS               *   102
```

```
no_space
  COMASM            *    317
  DC                *    704
  DISCINT           *    307
  GPIO              *    325
  HPIB              *    357
  RS                *    265
no_word
  COMASM            *    326
  DC                *    713
  DISCINT                316   725
  GPIO              *    334
  HPIB                   370   861
  RS                     274  1565
nodefault
  STRINT                 117   119
noisr
  POWERUP                411   432
noisrb
  POWERUP                436   438   448
nomatch
  MATCH                   84    88
non_dominant_support
  ASM_SCLIP         *    169
  ASM_STEXT         *    158
  DGL_AUTL          *    179
nopes
  FASTMOVE                18    27    36
normal_recover
  POWERUP                581   583   586
noroom
  ASM                    219   232
noroom
  DC                    2987  3000
nosgn
  STRINT                 154   156
nosign
  STRINT                 112   114
nospaces
  STRINT                 147   151
nosub
  EVALGVR                176   178
noswap
  SWAP8                   10    12
not125
  DC                    1171  1174
not_hpib
  COMASM            *    314
  DC                *    701
  DISCINT           *    304
  GPIO              *    322
  HPIB              *    364
  RS                *    262
not_isr
  RS                    2020  2030
not_lstn
  COMASM            *    328
  DC                *    715
  DISCINT                318   728
  GPIO              *    336
  HPIB                   378   863
  RS                *    276
not_special
  MODIV                   57    59    62
```

```
not_talk
  COMASM            *    327
  DC                *    714
  DISCINT                317   731
  GPIO              *    335
  HPIB                   377   865
  RS                *    275
notfound
  ROMCALL                187   205
notnil
  EVALGVR                158   168
null
  STRG1                  259   263
nxtdentry
  ROMCALL                189   195   199
nxtdsector
  ROMCALL                179   204
nxtext2
  MATCH                  103   105
nxtext3
  MATCH                  110   112
nzbit
  SETSTUFF                32    36
nzbyte
  SETSTUFF                18    26
nzword
  SETSTUFF                42    48
object
  EVALGVR                 49    69    70    71    85    92    96   103   106   113   115   119   121   128   130   149   150
offset
  MATCH                   14    91
oiisr1
  DC                    1636  1639
ok
  RSTRINT                 87    90
  SETSTUFF                55    58
  STRINT                 163   168
ok_xmit
  RS                *   1880
old_a5
  ASM_SCLIP         *    170
  ASM_STEXT         *    159
  DGL_AUTL          *    180
old_a6
  ASM_SCLIP         *    171
  ASM_STEXT         *    160
  DGL_AUTL          *    181
old66
  FMINIT            *    113
  GPIODVR           *     63
oldindex
  RSTRINT                 13    31    88
on_intr_int
  DC                     819  1632  1635
on_intr_isr
  DC                    1611  1631
op
  EVALGVR                 16   175
op_done
  BYE_DVR                122   193   352
op_poll
  BYE_DVR                123   669   760
```

```
r1018
  BUB_DVR              363   364
r1040
  BUB_DVR          *   365
r1048
  BUB_DVR              367   370
r1068
  BUB_DVR              369   382
r1124
  BUB_DVR              368   384
r1150
  BUB_DVR              386   389
r1162
  BUB_DVR              353   392
r1176
  BUB_DVR              329   395
r1332
  BUB_DVR              428   440
r140
  BUB_DVR              185   188   189
r1420
  BUB_DVR              465   468
r1428
  BUB_DVR              467   471
r1450
  BUB_DVR              472   476
r1456
  BUB_DVR              475   478
r1480
  BUB_DVR              485   487
r152
  BUB_DVR              190   193
r1734
  BUB_DVR              531   540
r1820
  BUB_DVR              572   575
r1830
  BUB_DVR              574   578
r1860
  BUB_DVR              579   581   585
r1876
  BUB_DVR              584   588
r1880
  BUB_DVR              594   596
r190
  BUB_DVR              194   197
r196
  BUB_DVR              198   199
r2132
  BUB_DVR              649   654
r2133
  BUB_DVR          *   653
r2135
  BUB_DVR              652   656
r2143
  BUB_DVR              650   653
r2196
  BUB_DVR              661   664
r2232
  BUB_DVR              646   667
r2240
  BUB_DVR              668   670   675
r2280
  BUB_DVR              676   677
```

```
r2334
  BUB_DVR              680   687
r2348
  BUB_DVR              672   695
r2350
  BUB_DVR              696   701
r2412
  BUB_DVR              697   703   706
r2440
  BUB_DVR              699   704   708
r2454
  BUB_DVR              710   711
r2550
  BUB_DVR          *   726
r2586
  BUB_DVR              728   730
r2664
  BUB_DVR              747   752
r2712
  BUB_DVR              748   750   754
r2716
  BUB_DVR              744   758
r2750
  BUB_DVR              759   763
r2752
  BUB_DVR              764   765
r2806
  BUB_DVR              768   774
r2820
  BUB_DVR              761   778
r2830
  BUB_DVR              779   780
r2926
  BUB_DVR          *   797
r2962
  BUB_DVR              799   801
r3008
  BUB_DVR              714   785   816
r356
  BUB_DVR              239   243
r366
  BUB_DVR              242   244
r700
  BUB_DVR              317   331
r76
  BUB_DVR              183   184
r800
  BUB_DVR              327   333
r922
  BUB_DVR              348   349
r944
  BUB_DVR              350   351
r_intIvl
  ASM                   75   118
radd
  GPIODVR          *    64
raddrec
  FORMAT               236   304
  VERIFY               156   233
rand
  RANDOM                15    65
random
  RANDOM                14    37
```

```
sysstate
  COMASM                346   440   482   891
  DC                    733   873  1181  1350  1925  2435  2511  2524  2564  2613  2898
  DISCINT               336   777
  FORMAT                 22   414   526
  GPIO                  354   562   594
  GPIODVR                92   140
  HPIB                  396   914   969
  POWERUP               723   728   748   752   777   798   869
  RS                    294  2068
  VERIFY                 19   382
sysglobals
  ALLOCATE                4     6
  ASM                    46    48    49    50    51    52
  COMASM                306   342   343
  DC                    693   729   730
  DISCINT               296   332   333
  EVALGVR                 5     7
  FMINIT                 36   151   152
  FORMAT                 16   572   573
  GPIO                  314   350   351
  GPIODVR                84   118
  HPIB                  356   392   393
  NEWWORDS                7    10
  POWERUP                15    23    24    25    26    27    28    29    30    31    32
  POWCALL                 7     9    10
  RS                    254   290   291
  RSTRINT                 2     4
  STRG1                  17    18
  STRG2                  17    18
  STRINT                  2     4
  SYSDEF                368   371
  VERIFY                 14   427   428
sysglobals_base
  SYSDEF                380   383
sysglobals_fsidc
  SYSDEF                372   375
sysglobals_sysglobals
  SYSDEF                376   379
systable
  SYSDEF          *      19
system_init
  DGL_AUTL        *      74
  DGL_IBODY              78    88
t_bw_off
  COMASM                273   782
  DC              *     660
  DISCINT               263  1356
  GPIO                  281   923   940  1007  1042  1086  1097  1128
  HPIB                  323  2078
  RS                    221  1530
t_dmapri
  COMASM                290   785
  DC              *     677
  DISCINT               280  1414
  GPIO            *     298
  HPIB            *     340
  RS              *     238
```

```
t_pm_off
  COMASM          *     289
  DC              *     676
  DISCINT         *     279
  GPIO            *     297
  HPIB            *     339
  RS              *     237
t_pr_off
  COMASM                286   575
  DC                    673  1784  1842
  DISCINT         *     276
  GPIO            *     294
  HPIB            *     336
  RS              *     234
t_sc_off
  COMASM                259   401   559   650
  DC                    646  1881
  DISCINT               249  1454
  GPIO                  267   950
  HPIB                  309  2204
  RS                    207  1629  1689  1705
t_si_off
  COMASM          *     288
  DC              *     675
  DISCINT         *     278
  GPIO            *     296
  HPIB            *     338
  RS              *     236
tact_off
  COMASM                260   402   530   560
  DC                    647  1882  1929  1948
  DISCINT               250  1401  1463  1470  1477
  GPIO                  268   906   961   967   995  1103
  HPIB                  310  2123  2138  2168  2211  2217  2285
  RS                    208  1571  1577  1626  1688
tb_a1
  BOOTDEFS        *      45
tb_a2
  BOOTDEFS        *      46
tb_auxpt
  BOOTDEFS        *      42
tb_bcd
  BOOTDEFS        *      43
tb_bin
  BOOTDEFS        *      44
tb_pwt
  BOOTDEFS        *      38
tb_pwt4
  BOOTDEFS        *      40
tb_pwt8
  BOOTDEFS        *      39
tb_pwtt
  BOOTDEFS        *      41
tbsz_off
  COMASM          *     283
  DC              *     670
  DISCINT         *     273
  GPIO            *     291
  HPIB            *     333
  RS              *     231
```

```
try_timer2
  DC                2633  2638
try_timer3
  DC                2635  2642
trymonitor
  POWERUP            192   617
trysend
  DC                2612  2616  2621  2624  2626  2641
tt_burst
  COMASM          *  296
  DC              *  683
  DISCINT            286  1151  1470
  GPIO               304   967
  HPIB               346  1681  2217
  RS              *  244
tt_dma
  COMASM             295   391
  DC              *  682
  DISCINT            285  1155  1401
  GPIO               303   906
  HPIB               345  1685  2123  2168
  RS              *  243
tt_empty_fifo
  DISCINT           1506  1519  1525  1529
tt_fhs
  COMASM          *  297
  DC              *  684
  DISCINT            287  1477
  GPIO               305   995  1124
  HPIB               347  2285
  RS                 245  1571  1626
tt_fifo_empty
  DISCINT           1527  1531
tt_int
  COMASM          *  294
  DC              *  681
  DISCINT            284  1153  1463
  GPIO               302   961  1103
  HPIB               344  1683  2138  2211
  RS                 242  1577
tt_non_sys
  DISCINT           1511  1521
tt_rts
  DISCINT           1503  1533
ttmp_off
  COMASM          *  258
  DC              *  645
  DISCINT         *  248
  GPIO            *  266
  HPIB            *  308
  RS              *  206
tusr_off
  COMASM          *  261
  DC                 648  1922
  DISCINT            251  1362  1451
  GPIO               269   865   947
  HPIB               311  2082  2201
  RS                 209  1533
tx_int
  DC                 818  1620  1865  2016
txbuff
  DC                 764   964  2976  3002  3007
txctrlbuffroom
  DC                1303  2738  2902  2941  2996
```

```
txdatabuffroom
  DC                1308  2746  2807  2983
txendblockspace
  DC                 762  2976  3002  3007
ubyte
  EVALGVR             59   118
ubyteerr
  EVALGVR             28   124
ucsdmodule
  SWAP8                3     4
umode
  POWERUP            359   363
un
  FMINIT             116   276   341   626   637
unclipped_draw
  ASM_SCLIP          204   295
  ASM_STEXT       *  193
  DGL_AUTL        *  214
unclipped_move
  ASM_SCLIP          205   233   281
  ASM_STEXT       *  194
  DGL_AUTL        *  215
uncorrect
  BUB_DVR            126   671
underscore
  RS                 512   578
unfix
  ROMCALL             59    72    91   164
unfix1
  ROMCALL             92    94
unin2
  GPIO              1098  1100
uninput
  GPIO              1081  1094
unit
  FORMAT              102   281   313   369   393   451
  VERIFY               95   210   242   272   323
unl
  DISCINT         *  593
  FORMAT               69   266   283   298   315   330   377   395   425   459   494
  HPIB                714  1597
  VERIFY               63   195   212   227   244   259   277   331   368
unout2
  GPIO              1087  1089
unoutput
  GPIO            *  1082
unpackit
  GASSM              104   117
unpackrow
  GASSM              108   113
unpackrow2
  GASSM              109   111
unpkkana
  GASSM               94   115
unpkroman
  GASSM               92   102
unt
  DISCINT         *  594
  FORMAT               68   265   297   329   426
  HPIB            *  715
  VERIFY               62   194   226   258
update
  ASM_STEXT          407   437   465
```

ALLOCATE

Description

ALLOCATE contains code to allocate a user-specifiable number of bytes off the stack.

```
PAGE   1 [3.0] 12/26/84 22:52:20  ASSEMBLY OF ALLOCATE.TEXT                                  *** File name: ALLOCATE ***

PASS 1 COMPLETE. ERRORS: 0
    1                           * FUNCTION ASM_ALLOCATE(SIZE: INTEGER): ANYPTR;
    2
    3                                   DEF     ASM_ALLOCATE
    4                                   REFA    SYSGLOBALS,STACKFUDGE
    5
    6                   FFFF FFF2 HEAPPOINTER   EQU     SYSGLOBALS-14
    7
    8                   0000 0000 RETURN  EQU A0
    9                   0000 0001 ADDRESS EQU A1
   10                   0000 0002 TEMP    EQU A2
   11
   12                   0000 0000 SIZE    EQU D0
   13
   14                   0000 0000 ASM_ALLOCATE    EQU *
   15         00000000 205F                MOVEA.L (SP)+,RETURN
   16         00000002 201F                MOVE.L  (SP)+,SIZE
   17         00000004 584F                ADDQ    #4,SP           POP SPACE FOR RETURNED VALUE
   18
   19         00000006 5280                ADDQ.L  #1,SIZE         ROUND UP TO EVEN NUMBER
   20         00000008 0880 0000           BCLR    #0,SIZE
   21         0000000C 4480                NEG.L   SIZE
   22         0000000E 6E14                BGT.S   OVERFLOW        DISALLOW NEGATIVE SIZES
   23
   24         00000010 43F7 0800           LEA     0(SP,SIZE.L),ADDRESS
   25
   26         00000014 45E9 0000           LEA     -STACKFUDGE(ADDRESS),TEMP
   27         00000018 B5ED FFF2           CMPA.L  HEAPPOINTER(A5),TEMP
   28         0000001C 6F06                BLE.S   OVERFLOW
   29
   30         0000001E 2E49                MOVEA.L ADDRESS,SP
   31         00000020 4851                PEA     (ADDRESS)
   32         00000022 4ED0                JMP     (RETURN)
   33
   34         00000024 42A7        OVERFLOW CLR.L  -(SP)           RETURN NIL
   35         00000026 4ED0                JMP     (RETURN)
   36
   37                                   NOSYMS
   38                                   END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

ALPHALIST

Description

ALPHALIST alphabetizes a list of symbols.

Usage

ALPHALIST is used by the linking loader.

```
PAGE   1 [3.0] 12/28/84 21:00:31  ASSEMBLY OF ALPHALIST.TEXT                                  *** File name: ALPHALIST ***

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 6552                  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 48F0 1800             lea     0(symtable,Joffset.l),Jstr      standard string comparison:
   49    00000024 47F0 0800             lea     0(symtable,Ioffset.l),Istr
   50    00000028 7C00                  MOVEQ   #0,COMP                                                 (rdq)
   51    0000002A 1C1B                  MOVE.B  (ISTR)+,COMP                                            (rdq)
   52    0000002C 3A06                  MOVE.W  COMP,LEN                                                (rdq)
   53    0000002E 4845                  SWAP    LEN                                                     (rdq)
   54    00000030 3A06                  MOVE.W  COMP,LEN                DUPLICATE LENGTH FOR LATER USE  (rdq)
   55    00000032 1C1C                  MOVE.B  (JSTR)+,COMP                                            (rdq)
   56    00000034 BA06                  cmp.b   COMP,len                compare lengths                 (rdq)
   57    00000036 6302                  bls.s   str_comp
   58    00000038 3A06                  MOVE.W  COMP,LEN                get minimum of two string lengths(rdq)
```

```
PAGE   2 [3.0] 12/28/84 21:00:31  ASSEMBLY OF ALPHALIST.TEXT                                  *** File name: ALPHALIST ***

   59    0000003A 5305          str_comp subq.b #1,len                  loop if at least one character
   60    0000003C 6508                  bcs.s   cmp_end
   61    0000003E B70C          cmp_lp  cmpm.b  (Jstr)+,(Istr)+         compare string bodies
   62    00000040 5ECD FFFC             dbne    len,cmp_lp              loop till not equal or end of string
   63    00000044 6604                  bne.s   cmp                     if string bodies are equal,
   64    00000046 4845          cmp_end SWAP    LEN                             then compare lengths    (rdq)
   65    00000048 BA06                  CMP.B   COMP,LEN                                                (rdq)
   66    0000004A 640C          cmp     bcc.s   found                   if string(I) >= string(J), J loop done
   67
   68    0000004C 3541 0002             move.w  Joffset,2(J)            list[J+1] := joffset
   69    00000050 51CB FFCC             dbra    Jcount,J_loop           loop until J = 0
   70    00000054 3480                  move.w  Ioffset,(J)             if J = 0 then list[0] := ioffset
   71    00000056 6C04                  bra.s   next_I
   72
   73    00000058 3540 0002     found   move.w  Ioffset,2(J)            list[J+1] := ioffset
   74    0000005C 51CC FFB8     next_I  dbra    N,I_loop                loop until N = 0
   75
   76    00000060 4E75          done    rts                             wasn't that easy?
   77
   78                                   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		00000002
CMP	REL	66		0000004A
CMP_END	REL	64		00000046
CMP_LP	REL	61		0000003E
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
DFC	STREG	0		00000001
DONE	REL	76		00000060
FOUND	REL	73		00000058
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	74		0000005C
SFC	STREG	0		00000000
SP	AREG	0		00000007
SR	STREG	0		00000006
STR_COMP	REL	59		0000003A
SYMTABLE	AREG	14		00000000
USP	STREG	0		00000007
VBR	STREG	0		0000000A

Description

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.

```
PAGE   1 [3.0] 12/06/84 20:59:10  ASSEMBLY OF ASM.TEXT                                   *** File name: ASM ***

PASS 1 COMPLETE. ERRORS: 0
    1                       *
    2                       * CHANGES MADE FOR REV 3.0 AUG 83  rdq
    3                       *
    4                               MNAME   ASM
    5                               SPRINT
    6
    7                               DEF     ASM_ASM,ASM_POWERUP,ASM_ERRMSG,G_DOLLAR,STACKFUDGE
    8                               DEF     ASM_USERPROGRAM,ASM_CI_SWITCH,ASM_FINDROMS,ASM_CPYMSG
    9                               DEF     M68KTYPE,MSYSFLAGS                                      (rdq)
   10
   11                       SRC MODULE ASM;
   12                       SRC IMPORT SYSGLOBALS;
   13                       SRC EXPORT
   14                       SRC TYPE
   15                       SRC STRINGMAX = STRING[255];
   16                       SRC   PROCEDURE MOVELEFT (ANYVAR S,D: INTEGER; Z:INTEGER);
   17                       SRC   PROCEDURE MOVERIGHT(ANYVAR S,D: INTEGER; Z:INTEGER);
   18                       SRC   PROCEDURE FASTMOVE (       S,D: ANYPTR;  Z:INTEGER);
   19                       SRC   PROCEDURE NEWBYTES(VAR P: ANYPTR;        Z:INTEGER);
   20                       SRC   PROCEDURE POWERUP;
   21                       SRC   PROCEDURE ERRMSG;
   22                       SRC   PROCEDURE FINDROMS;
   23                       SRC   PROCEDURE F_PWR_ON;
   24                       SRC   PROCEDURE FLPYREAD(SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   25                       SRC   PROCEDURE FLPY_WRT(SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   26                       SRC   PROCEDURE FLPYMREAD(SECTOR_COUNT, SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   27                       SRC   PROCEDURE FLPYMWRITE(SECTOR_COUNT, SECTOR: INTEGER;  ANYVAR BUFFER: INTEGER);
   28                       SRC   PROCEDURE FLPYINIT(PTR: ANYPTR; I: SHORTINT);
   29                       SRC   PROCEDURE SETINTLEVEL(LEVEL: INTEGER);
   30                       SRC   FUNCTION  INTLEVEL:  INTEGER;
   31                       SRC   PROCEDURE NEWWORDS(VAR P: ANYPTR; WORDSIZE: INTEGER);
   32                       SRC   PROCEDURE USERPROGRAM (EXECLOC,INITSP: INTEGER);
   33                       SRC   PROCEDURE SAPPEND(VAR DEST: STRING;SRC:STRINGMAX);
   34                       SRC   FUNCTION IAND(A,B: INTEGER): INTEGER;
   35                       SRC   FUNCTION IOR (A,B: INTEGER): INTEGER;
   36                       SRC   PROCEDURE CI_SWITCH;
   37                       SRC   PROCEDURE INITVECTS;
   38                       SRC   PROCEDURE CPYMSG(MSG: STRING255);
   39                       SRC   FUNCTION MEMAVAIL:INTEGER;
   40                       SRC   FUNCTION TICKER:INTEGER;
   41                       SRC   PROCEDURE CACHE_ON;
   42                       SRC   PROCEDURE CACHE_OFF;
   43                       SRC END;
   44
   45                              REFR    FS_FWRITESTRINT,INITLOAD_INITLOAD
   46                              REFA    SYSGLOBALS,LOADER
   47
   48              FFFF FFFE ESCAPECODE    EQU SYSGLOBALS-2
   49              FFFF FFF6 RECOVERBLOCK  EQU SYSGLOBALS-10
   50              FFFF FFF2 HEAPPOINTER   EQU SYSGLOBALS-14
   51              FFFF FFEE HEAPBASE      EQU SYSGLOBALS-18
   52              FFFF FFEA IORESULT      EQU SYSGLOBALS-22
   53              FFFF FFBA SYSDEFS       EQU LOADER-70           LINKED LIST OF PERMANT PROGRAMS
   54
   55              0000 1388 SUPSTACKSPACE EQU 5000        SPACE FOR SUPERVISOR STACK
   56              0000 01F4 STACKFUDGE    EQU 500         SLOP SPACE FOR STACK OVERFLOW CHECK
   57
   58                      *       MAGIC NUMBERS, CONSULT 'POWERUP' LISTING, DEBUGGER and INITBUG  (rdq)
```

```
PAGE   2 [3.0] 12/06/84 20:59:10  ASSEMBLY OF ASM.TEXT                                   *** File name: ASM ***

   59              FFFF FB70 INITSTACK     EQU $FFFFFB70                                           (rdq)
   60              FFFF FB74 INITPC        EQU INITSTACK+4                                         (rdq)
   61              FFFF FB78 INITRECOVER   EQU INITPC+4                                            (rdq)
   62              FFFF FB7C G_DOLLAR      EQU INITRECOVER+4                                       (rdq)
   63              FFFF FB96 INITSR        EQU $FFFFFB96                                           (rdq)
   64              FFFF FB98 M68KTYPE      EQU INITSR+2    PROCESSOR TYPE BYTE 0=68000,else 680xx  (rdq)
   65              FFFF FB99 MSYSFLAGS     EQU M68KTYPE+1  MORE SYSFLAGS BIT 0 = CACHE PRESENT     (rdq)
   66
   67              FFFF FDCE LOWMEM        EQU $FFFFFDCE   BOOT ROM DEFINED ADDRESSES
   68              FFFF FB00 HIGHMEM       EQU $FFFFFB00   LEAVES ROOM FOR VECTORS, MONITOR STUFF, ETC.
   69              FFFF FF52 TRAP11        EQU $FFFFFF52
   70              FFFF FF58 TRAP10        EQU $FFFFFF58
   71              FFFF FF94 TRAP0         EQU $FFFFFF94
   72
   73              FFFF FED4 F_AREA        EQU $FFFFFED4   BOOTROM 3.0 MAGIC CONSTANTS TO    JS 9/12/83
   74              0000 009A REMOTE_ADDR   EQU $9A         ALLOW US TO TURN OFF INTERRUPTS   JS 9/12/83
   75              0000 0003 R_INTLVL      EQU 3           IF BOOTING FROM REMOTE INTERFACE  JS 9/12/83
   76              0000 3FFE BOOT_ID       EQU $3FFE                                         JS 9/12/83
   77              0080 0000 NO_CARD       EQU $800000                                       JS 9/12/83
   78
   79              0000 0150 CRTMSG        EQU $150
   80                        *CRTCLEAR      EQU $148        CAN'T USE DUE TO BOOT ROM BUG
   81              FFFF FED2 SYSFLAG       EQU -302
   82
   83              0000 4EF9 JMP           EQU $4EF9
   84              0000 0004 RETURN        EQU A4
   85
   86
   87
   88
   89              0000 0000 LOWCODE EQU *                         LOWEST CODE LOCATION
   90              0010 0000 HIGHCODE EQU *+$100000                1 MEGABYTE MAX ROM AREA
   91
   92                      * NOTE: INITLOAD_INITLOAD  FIRST INSTRUCTION IS JSR ASM_POWERUP
   93    00000000 F0               DC.B    $F0,$FF,'P',1           ROM HEADER
   94    00000004 0000 0000        DC.L    0,INITLOAD_INITLOAD-*   CHECKSUM, EXECUTION ADDRESS
   95    0000000C 00               DC.B    0,0,0,0,0,0             IDENTIFICATION JUNK
   96
   97                      ******  ESTABLISH MINIMUM ENVIRONMENT FOR POWERUP LOAD OPERATIONS ********
   98              0000 0012 ASM_POWERUP   EQU *
   99    00000012 285F             MOVEA.L (SP)+,RETURN
  100    00000014 4FFA FFEA        LEA     LOWCODE,SP              SOFT SYSTEM STACK = JUST BELOW CODE
  101    00000018 FFF8 FDCE        CMPA.L  LOWMEM,SP               TEST FOR HARD OR SOFT SYSTEM
  102    0000001C 6204             BHI.S   SOFT                    SOFT IF SP > LOWMEM^
  103    0000001E 4FF8 FB00        LEA     HIGHMEM,SP              HARD SYSTEM STACK = HIGH MEMEMORY
  104              0000 0022 SOFT  EQU *
  105
  106    00000022 4BEF EC78        LEA     -SUPSTACKSPACE(SP),A5   ALLOCATE OPERATING SYSTEM STACK
  107    00000026 4BED 8000        LEA     -32768(A5),A5           SYSTEM GLOBALS START AT 32766(A5)
  108    0000002A 21CD FB7C        MOVE.L  A5,G_DOLLAR             SAVE FOR ISR's
  109
  110    0000002E 2B78 FDCE        MOVE.L  LOWMEM,HEAPPOINTER(A5)  INITIALIZE HEAP
                  FFF2
  111
  112    00000034 0C78 0003        CMPI.W  #3,BOOT_ID              3.0 BOOTROM ?         JS 9/12/83
                  3FFE
  113    0000003A 6614             BNE.S   SKIP                    IF NOT THEN SKIP      JS 9/12/83
```

```
  114  00000C3C 2078 FED4          MOVEA.L F_APEA,A0              GET STOLEN RAM PTR    JS 9/12/83
  115  00000C40 2068 009A          MOVEA.L REMOTE_ADDR(A0),A0     SEE IF REMOTE BOOT    JS 9/12/83
  116  00000C44 B1FC 0080          CMPA.L  #NO_CARD,A0                                  JS 9/12/83
                0000
  117  00000C4A 6704               BEQ.S   SKIP                   IF NOT THEN SKIP      JS 9/12/83
  118  00000C4C 4228 0003          CLR.B   R_INTLVL(A0)           ELSE CLEAR INTERRUPTS JS 9/12/83
  119
  120  00000C50 6100 01C4 SKIP     BSR     WHICH_PROCESSOR        FINDOUT IF 68000 OR OTHER       (rdq)
  121
  122  00000054 46FC 2000          MOVE    #$2000,SR              ENABLE INTERRUPTS
  123
  124  00000058 41F8 FF52          LEA     TRAP11,A0              SUPERVISOR CALL TRAP
  125  0000005C 30FC 4EF9          MOVE    #JMP,(A0)+
  126  00000060 43FA 0026          LEA     SUPERCALL,A1
  127  00000064 20C9               MOVE.L  A1,(A0)+
  128
  129                       *      LEA     TRAP10,A0              ESCAPE(N) TRAP
  130  00000066 30FC 4EF9          MOVE    #JMP,(A0)+
  131  0000006A 43FA 0016          LEA     ESCN,A1
  132  0000006E 20C9               MOVE.L  A1,(A0)+
  133
  134  00000070 41F8 FF94          LEA     TRAP0,A0               COMPILED LINE TRAP
  135  00000074 30FC 4EF9          MOVE    #JMP,(A0)+
  136  00000078 43FA 0022          LEA     P_BREAK,A1
  137  0000007C 20C9               MOVE.L  A1,(A0)+
  138
  139  0000007E 4ED4               JMP     (RETURN)               ALL DONE
  140
  141  00000080 4E75       ASM_ASM RTS                    ASM MODULE 'BODY'
  142
  143                             ****************  TEMPORARY TRAP 10 HANDLER **********************************
  144  00000082 2E6D FFF6 ESCN     MOVEA.L RECOVERBLOCK(A5),SP
  145  00000086 4E75               RTS
  146
  147                             ********(rdq)****  PERMANENT TRAP 11 HANDLER **********************************
  148            0000 0088 SUPERCALL EQU *                  TRAP #11, GETS INTO SUPERVISOR MODE, SAVES SR
  149  00000088 4A38 FB98          TST.B   M68KTYPE
  150  0000008C 6608               BNE.S   SUPERCALL_1
  151  0000008E 3F17               MOVE.W  (SP),-(SP)     MOVE STATUS REGISTER DOWN
  152  00000090 2F6F 0004          MOVE.L  4(SP),2(SP)    MOVE RETURN ADDRESS DOWN
                0002
  153            0000 0096 SUPERCALL_1 EQU *
  154  00000096 3F5F 0004          MOVE.W  (SP)+,4(SP)    MOVE STATUS REGISTER UP
  155  0000009A 4E75               RTS                    RETURN TO CALLER IN SUPERVISOR MODE
  156
  157                             ****************  TEMPORARY TRAP 0 HANDLER, DISCARDS LINE NO.s ************
  158  0000009C 54AF 0002 P_BREAK  ADDQ.L  #2,2(SP)
  159  000000A0 4E73               RTE
  160
  161                             ****************  FINISH ENVIRONMENT SETUP THEN CALL USER PROGRAM **********
  162            0000 00A2 ASM_USERPROGRAM EQU     *        PROC USERPROGRAM(ENTRYPOINT,INITSTACK:INTEGER)
  163  000000A2 265F               MOVEA.L (SP)+,A3       RETURN ADDRESS
  164  000000A4 205F               MOVEA.L (SP)+,A0       INITIAL SP
  165  000000A6 225F               MOVEA.L (SP)+,A1       INITIAL PC
  166  000000A8 2F0B               MOVE.L  A3,-(SP)       RESTORE RETURN
  167
  168  000000AA 45E8 FE0C          LEA     -STACKFUDGE(A0),A2
```



```
  169  000000AE B5ED FFF2          CMPA.L  HEAPPOINTER(A5),A2       COMPARE
  170  000000B2 6202               BHI.S   *+4                    TEST
  171  000000B4 4E42               TRAP    #2                     STACK OVERFLOW
  172
  173  000000B6 2F2D FFF6          MOVE.L  RECOVERBLOCK(A5),-(SP)  TRY
  174  000000BA 2F0E               MOVE.L  A6,-(SP)               SAVE A6
  175  000000BC 487A 002E          PEA     USER_RECOVER           RECOVER CODE
  176  000000C0 2B4F FFF6          MOVE.L  SP,RECOVERBLOCK(A5)
  177
  178  000000C4 21CF FB78          MOVE.L  SP,INITRECOVER         SAVE FOR DEBUGGER
  179  000000C8 21C9 FB74          MOVE.L  A1,INITPC              SAVE FOR DEBUGGER
  180
  181  000000CC 4DD5               LEA     (A5),A6                FRAME POINTER
  182  000000CE 4E60               MOVE    A0,USP                 SET UP STACK
  183  000000D0 46FC 0000          MOVE    #$0000,SR              GO INTO USER MODE
  184
  185  000000D4 45EF FFF8          LEA     -8(SP),A2              SPACE FOR RETURN ADDRESS, DYNAMIC LINK
  186  000000D8 21CA FB70          MOVE.L  A2,INITSTACK           SAVE NON-LOCAL GOTO
  187  000000DC 4278 FB96          CLR.W   INITSR                 FAKE SR FOR STOP KEY                  (rdq)
  188  000000E0 42AD FFEE          CLR.L   HEAPBASE(A5)           FOR MEMORY MANAGER
  189
  190  000000E4 4E91               JSR     (A1)                   CALL USER PROGRAM
  191
  192  000000E6 426D FFFE          CLR.W   ESCAPECODE(A5)         NORMAL EXIT
  193  000000EA 4E4A               TRAP    #10                    ESCAPE(0), DONE TO CLOSE FILES
  194
  195            0000 00EC USER_RECOVER    EQU *
  196  000000EC 2C5F               MOVEA.L (SP)+,A6               RESTORE A6
  197  000000EE 2B5F FFF6          MOVE.L  (SP)+,RECOVERBLOCK(A5)  RESTORE RECOVER BLOCK
  198  000000F2 204F               MOVEA.L SP,A0                  SAVE STACK POINTER
  199  000000F4 4E4B               TRAP    #11                    GET INTO SUPERVISOR MODE
  200  000000F6 2E48               MOVEA.L A0,SP                  RESTORE STACK POINTER
  201  000000F8 4E75               RTS
  202
  203                             ****************  SWITCH TO SUPERVISOR MODE & STACK ************************
  204            0000 00FA ASM_CI_SWITCH EQU *
  205  000000FA 205F               MOVEA.L (SP)+,A0           RETURN ADDRESS
  206  000000FC 4E4B               TRAP    #11                GET ONTO SUPERVISOR STACK
  207  000000FE 544F               ADDQ    #2,SP              DISCARD SR
  208  00000100 4ED0               JMP     (A0)               DONE
  209
  210                             ****************  HOOK ROM'C MODULES INTO SYSDEFS **************************
  211            0000 4000 K16    EQU     $4000              16 K INCREMENTS
  212            0000 0012 MDLINK EQU     18                 OFFSET OF MODULE DESCRIPTOR
  213            0000 0028 MDSIZE EQU     40                 SIZE OF DESCRIPTOR
  214            0000 0102 ASM_FINDROMS EQU *
  215  00000102 41F9 0002          LEA     $20000,A0                BEGINNING OF SEARCH AREA
                0000
  216  00000108 226D FFF2          MOVEA.L HEAPPOINTER(A5),A1       HEAP START
  217  0000010C 202D FFBA          MOVE.L  SYSDEFS(A5),D0           LINKED LIST HEAD
  218  00000110 0C90 F0FF SEARCH   CMPI.L  #$F0FF5080,(A0)          PASCAL OPTION?
                5080
  219  00000116 6624               BNE.S   NOROM                  NO MATCH
  220  00000118 45E8 0012          LEA     MDLINK(A0),A2          LOCATE DESCRIPTOR
  221
  222  0000011C 4CEA 18FE COPY     MOVEM.L 4(A2),D1-D7/A3-A4      REMAINDER OF MDB
                0004
```

```
PAGE   5 [3.0] 12/26/84 20:59:10  ASSEMBLY OF ASM.TEXT                                   *** File name: ASM ***

  223    00000122 48D1 18FF          MOVEM.L D0-D7/A3-A4,(A1)        COPY ONTO HEAP
  224    00000126 2009               MOVE.L  A1,D0                   INSERT LINK INTO LIST
  225    00000128 D3FC 0000          ADDA.L  #MDSIZE,A1              ALLOCATE FROM HEAP
                  0028
  226    0000012E 2B49 FFF2          MOVE.L  A1,HEAPPOINTER(A5)      RESTORE HEAP
  227    00000132 2B40 FFBA          MOVE.L  D0,SYSDEFS(A5)          RESTORE LIST
  228
  229    00000136 2452               MOVEA.L (A2),A2         MOVE DOWN CHAIN
  230    00000138 220A               MOVE.L  A2,D1           TEST FOR NIL
  231    0000013A 66EC               BNE.S   COPY            NOT END OF CHAIN
  232    0000013C D0FC 4000 NOROM    ADDA.W  #K16,A0         NEXT ROM BOUNDARY
  233    00000140 B1FC 0020          CMPA.L  #$200000,A0     END OF ROM AREA?
                  0000
  234    00000146 66C8               BNE.S   SEARCH          NOPE
  235    00000148 4E75       FINDXIT RTS
  236
  237                        ****************  CRT CLEAR ROUTINE ****************************************
  238              0000 014A CRTCLEAR        EQU *
  239    0000014A 4E56 FFAE          LINK    A6,#-82
  240    0000014E 41D7               LEA     (SP),A0
  241    00000150 704F               MOVEQ   #79,D0
  242    00000152 0838 0000          BTST    #0,SYSFLAG      TEST FOR 50 CHAR SCREEN
                  FED2
  243    00000158 6702               BEQ.S   LC1
  244    0000015A 7031               MOVEQ   #49,D0
  245    0000015C 10FC 0020 LC1      MOVE.B  #' ',(A0)+
  246    00000160 51C8 FFFA          DBRA    D0,LC1
  247    00000164 4210               CLR.B   (A0)
  248    00000166 422E FFFF          CLR.B   -1(A6)
  249    0000016A 41D7       LC2     LEA     (SP),A0
  250    0000016C 4240               CLR     D0
  251    0000016E 102E FFFF          MOVE.B  -1(A6),D0
  252    00000172 4EB8 0150          JSR     CRTMSG
  253    00000176 522E FFFF          ADDQ.B  #1,-1(A6)
  254    0000017A 002E 0018          CMPI.B  #24,-1(A6)
                  FFFF
  255    00000180 66E8               BNE.S   LC2
  256    00000182 4E5E               UNLK    A6
  257    00000184 4E75               RTS
  258
  259                        ****************  INTRODUCTORY MESSAGE ROUTINE  ****************************
  260              0000 0186 ASM_CPYMSG      EQU *
  261    00000186 4EBA FFC2          JSR     CRTCLEAR
  262    0000018A 7000               MOVEQ   #0,D0
  263    0000018C 41FA 00A6          LEA     M,A0                    COPYRIGHT NOTICE
  264    00000190 4EB8 0150          JSR     CRTMSG
  265    00000194 7002               MOVEQ   #2,D0
  266    00000196 225F               MOVEA.L (SP)+,A1
  267    00000198 205F               MOVEA.L (SP)+,A0
  268    0000019A 4241               CLR     D1
  269    0000019C 1218               MOVE.B  (A0)+,D1
  270    0000019E 4230 1000          CLR.B   0(A0,D1)
  271    000001A2 2F09               MOVE.L  A1,-(SP)
  272    000001A4 4EB8 0150          JSR     CRTMSG
  273    000001A8 4E75               RTS
  274
  275                        ****************  PRIMITIVE ERROR MESSAGE ROUTINE  *************************
```

```
PAGE   6 [3.0] 12/26/84 20:59:10  ASSEMBLY OF ASM.TEXT                                   *** File name: ASM ***

  276              FFFF FFAA STR     EQU -86
  277              FFFF FFFC I       EQU -4
  278
  279    000001AA 4281       INT     CLR.L   D1
  280    000001AC 122E FFAA          MOVE.B  STR(A6),D1
  281    000001B0 5281               ADDQ.L  #1,D1
  282    000001B2 2D41 FFFC          MOVE.L  D1,I(A6)
  283    000001B6 1F3C 0050          MOVE.B  #80,-(SP)
  284    000001BA 486E FFAA          PEA     STR(A6)
  285    000001BE 486E FFFC          PEA     I(A6)
  286    000001C2 2F00               MOVE.L  D0,-(SP)
  287    000001C4 3F3C FFFF          MOVE.W  #-1,-(SP)
  288    000001C8 4EBA FE36          JSR     FS_FWRITESTRINT
  289    000001CC 4E75               RTS
  290
  291              0000 01CE ASM_ERRMSG  EQU *
  292    000001CE 4E4B               TRAP    #11
  293    000001D0 4E56 FFAA          LINK    A6,#STR
  294    000001D4 4EBA FF74          JSR     CRTCLEAR
  295    000001D8 48FA 003E          MOVEM.L M1,D1-D5
                  0080
  296    000001DE 48EE 003E          MOVEM.L D1-D5,STR(A6)
                  FFAA
  297    000001E4 202D FFEA          MOVE.L  IORESULT(A5),D0
  298    000001E8 61C0               BSR.S   INT
  299    000001EA 302D FFFE          MOVE.W  ESCAPECODE(A5),D0
  300    000001EE 48C0               EXT.L   D0
  301    000001F0 61B8               BSR.S   INT
  302    000001F2 4280               CLR.L   D0
  303    000001F4 41EE FFAA          LEA     STR(A6),A0
  304    000001F8 4241               CLR     D1
  305    000001FA 1210               MOVE.B  (A0),D1
  306    000001FC 4230 1001          CLR.B   1(A0,D1)
  307    00000200 5248               ADDQ    #1,A0
  308    00000202 4EB8 0150          JSR     CRTMSG
  309
  310    00000206 701C               MOVEQ   #28,D0
  311    00000208 51C9 FFFE L        DBRA    D1,L
  312    0000020C 51C8 FFFA          DBRA    D0,L
  313
  314    00000210 4E5E               UNLK    A6
  315    00000212 46DF               MOVE    (SP)+,SR
  316    00000214 4E75               RTS
  317
  318                        ******(rdg)******  FIND OUT WHICH PROCESSOR NOW RUNNING **********************
  319              0000 0216 WHICH_PROCESSOR EQU *
  320    00000216 41F8 FFS2          LEA     TRAP11,A0       SETUP TRAP11
  321    0000021A 30FC 4EF9          MOVE.W  #JMP,(A0)+      THIS SETUP WILL BE DISCARDED
  322    0000021E 20BC 0000          MOVE.L  #TRAP_SERV,(A0)
                  0230
  323    00000224 200F               MOVE.L  SP,D0           SAVE STACK LOCATION
  324    00000226 4E4B               TRAP    #11             FIND OUT HOW MANY BYTES PUSHED ON STACK
  325    00000228 5D80               SUBQ.L  #6,D0
  326    0000022A 56F8 FB98          SNE     M68KTYPE        =0 is 68000 <>0 is 680xx
  327    0000022E 4E75               RTS
  328              0000 0230 TRAP_SERV       EQU *
  329    00000230 908F               SUB.L   SP,D0
```

```
PAGE   7 [3.0] 12/26/84 20:59:10  ASSEMBLY OF ASM.TEXT                                  *** File name: ASM ***

  330    00000232 4E73              RTE
  331
  332                       *-------------------------------------------------------
  333    00000234 43        M       DC.B    'Copyright 1984 Hewlett-Packard Company.',0
  334    0000025C 12        M1      DC.B    18,'10RESULT, ERROR = '
  335
  336
  337                               NOSYMS
  338                               END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

ASM_SCLIP

Description

ASM_SCLIP contains software clipping routines for DGL.

Usage

Called as a DGL "tool."

```
PASS 1 COMPLETE. ERRORS: 0
    1                       *  Graphics Low End
    2                       *
    3                       *  Module     = Software clipping
    4                       *  Programer  = BJS
    5                       *  Date       = 10-5-82
    6                       *
    7                       *  Purpose : To provide software clipping routines.
    8                       *
    9                       *  Rev history
   10                       *
   11                       *    Created    - 10- 5-82
   12                       *    Modified   - 11-23-82    Removed test for static links on proc calls
   13                       *
   14                       *
   15                       *     (c) Copyright Hewlett-Packard Company, 1983.
   16                       *     All rights are reserved.  Copying or other
   17                       *     reproduction of this program except for archival
   18                       *     purposes is prohibited without the prior
   19                       *     written consent of Hewlett-Packard Company.
   20                       *
   21                       *
   22                       *                    RESTRICTED RIGHTS LEGEND
   23                       *
   24                       *     Use, duplication, or disclosure by the Government
   25                       *     is subject to restrictions as set forth in
   26                       *     paragraph (b) (3) (B) of the Rights in Technical
   27                       *     Data and Computer Software clause in
   28                       *     DAR 7-104.9(a).
   29                       *
   30                       *     HEWLETT-PACKARD COMPANY
   31                       *     Fort Collins, Colorado
   32                       *
   33                       *
   34                             mname GLE_ASCLIP
   35
   36                               src module GLE_ASCLIP;
   37                               src import GLE_TYPES;
   38                               src export
   39                               src procedure gle_soft_clip_move ( gcb : graphics_control_block_ptr );
   40                               src procedure gle_soft_clip_draw ( gcb : graphics_control_block_ptr );
   41                               src end;
   42
   43
   44                             nosyms
   45                       *
   46                       *
   47                       *  Define entry points
   48                       *
   49 00000000                   rorg 0
   50
   51                             def GLE_ASCLIP_GLE_SOFT_CLIP_MOVE
   52                             def GLE_ASCLIP_GLE_SOFT_CLIP_DRAW
   53                             def GLE_ASCLIP_CLIPPING
   54                             def GLE_ASCLIP_GLE_ASCLIP
   55                       *
   56                       *    Set up globals
   57                       *
   58             0000 0004 GCB equ a4
```



```
   59
   60             0000 0004 X0 equ d4
   61             0000 0005 Y0 equ d5
   62             0000 0006 X1 equ d6
   63             0000 0007 Y1 equ d7
   64             0000 0000 clip_xmin equ a0
   65             0000 0001 clip_xmax equ a1
   66             0000 0002 clip_ymin equ a2
   67             0000 0003 clip_ymax equ a3
   68
   69                             INCLUDE ASM_TYPES
   70             0000 0018 AWAIT_BLANKING equ 24
   71             0000 018E BACKGROUND equ 398
   72             0000 0020 BUFFER_MODE equ 32
   73             0000 029C CALC_SOFT_TEXT_XFORM equ 668
   74             0000 029A CALC_TEXT_XFORM equ 666
   75             0000 01D2 CHAR_HEIGHT equ 466
   76             0000 01E6 CHAR_JUST_X equ 486
   77             0000 01EA CHAR_JUST_Y equ 490
   78             0000 0028 CHAR_SIZE equ 40
   79             0000 01B0 CHAR_SIZES equ 432
   80             0000 01D6 CHAR_SPACE equ 470
   81             0000 01CE CHAR_WIDTH equ 462
   82             0000 0030 CLEAR equ 48
   83             0000 0038 CLIP_LIMITS equ 56
   84             0000 01F2 CLIP_LIMITS_XMAX equ 498
   85             0000 01EE CLIP_LIMITS_XMIN equ 494
   86             0000 01FA CLIP_LIMITS_YMAX equ 506
   87             0000 01F6 CLIP_LIMITS_YMIN equ 502
   88             0000 0196 COLOR_MAP_SUPPORT equ 406
   89             0000 0190 COMPLEMENT_SUPPORT equ 400
   90             0000 01AA CONT_LINESTYLES equ 426
   91             0000 023A COSX_TABLE equ 570
   92             0000 024A COSY_TABLE equ 586
   93             0000 0208 CURRENT_BUFFER_MODE equ 520
   94             0000 0212 CURRENT_COLOR_INDEX equ 530
   95             0000 01FE CURRENT_CURSOR_STATE equ 510
   96             0000 0200 CURRENT_CURSOR_X equ 512
   97             0000 0204 CURRENT_CURSOR_Y equ 516
   98             0000 0216 CURRENT_DRAWING_MODE equ 534
   99             0000 0214 CURRENT_FILL_INDEX equ 532
  100             0000 020A CURRENT_LINESTYLE equ 522
  101             0000 0210 CURRENT_LINESTYLE_MODE equ 528
  102             0000 020C CURRENT_LINESTYLE_PATTERN equ 524
  103             0000 0218 CURRENT_LINEWIDTH equ 536
  104             0000 020E CURRENT_PATTERN_LENGTH equ 526
  105             0000 0220 CURRENT_POLYGON_BLUE equ 544
  106             0000 021A CURRENT_POLYGON_COLOR equ 538
  107             0000 021E CURRENT_POLYGON_GREEN equ 542
  108             0000 021C CURRENT_POLYGON_RED equ 540
  109             0000 01B2 CURRENT_POS_X equ 434
  110             0000 01B6 CURRENT_POS_Y equ 438
  111             0000 0040 CURSOR equ 64
  112             0000 0048 DEFINE_COLOR_MAP equ 72
  113             0000 0050 DEFINE_DRAWING_MODE equ 80
  114             0000 014C DEVICE_BUF equ 332
  115             0000 0154 DEVICE_INFO equ 340
```

```
                0000 0158 DEVICE_INFO_CHAR_COUNT equ 344
                0000 0150 DEV_DEP_STUFF equ 336
                0000 016C DISPLAY_HANDLER_CHAR_COUNT equ 364
                0000 0166 DISPLAY_HANDLER_NAME equ 358
                0000 0186 DISPLAY_MAX_X equ 390
                0000 018A DISPLAY_MAX_Y equ 394
                0000 017E DISPLAY_MIN_X equ 382
                0000 0182 DISPLAY_MIN_Y equ 386
                0000 015E DISPLAY_NAME equ 350
                0000 0164 DISPLAY_NAME_CHAR_COUNT equ 356
                0000 016E DISPLAY_RES_X equ 366
                0000 0176 DISPLAY_RES_Y equ 374
                0000 01A0 DITHER_SUPPORT equ 416
                0000 0058 DRAW equ 88
                0000 0118 DUMMY_XXX equ 280
                0000 01BA END_X equ 442
                0000 01BE END_Y equ 446
                0000 0194 ERASE_SUPPORT equ 404
                0000 015A ERROR_RETURN equ 346
                0000 C060 FILL_INDEX_COLOR equ 96
                0000 C068 FLUSH_BUFFER equ 104
                0000 01A6 GAMUT equ 422
                0000 C070 GET_COLOR_MAP equ 112
                0000 C080 GET_POLYGON_INFO equ 128
                0000 0078 GET_RASTER equ 120
                0000 0088 GLOAD equ 136
                0000 0090 GRAPHICS_ON_OFF equ 144
                0000 0098 GSTORE equ 152
                0000 00A0 INDEX_COLOR equ 160
                0000 0000 INF01 equ 0
                0000 0004 INF02 equ 4
                0000 0008 INF03 equ 8
                0000 000C INF04 equ 12
                0000 0010 INFO_PTR1 equ 16
                0000 0014 INFO_PTR2 equ 20
                0000 00A8 INQ_P1P2 equ 168
                0000 0148 IOCB equ 328
                0000 0120 IO_INQ_TIMEOUT equ 288
                0000 0128 IO_READ equ 296
                0000 0130 IO_SET_TIMEOUT equ 304
                0000 0138 IO_TERM equ 312
                0000 0140 IO_WRITE equ 320
                0000 01CC KATA equ 460
                0000 00B8 LINESTYLE equ 184
                0000 00B0 LINEWIDTH equ 176
                0000 01AE LINEWIDTHS equ 430
                0000 01DA LINE_SPACE equ 474
                0000 00C0 MARKER equ 192
                0000 01C8 MARKER_HEIGHT equ 456
                0000 00C8 MARKER_SIZE equ 200
                0000 01C2 MARKER_TYPE equ 450
                0000 01C4 MARKER_WIDTH equ 452
                0000 00D0 MOVE equ 208
                0000 0192 NON_DOMINANT_SUPPORT equ 402
                0000 0222 OLD_A5 equ 546
                0000 0226 OLD_A6 equ 550
                0000 00D8 OUTPUT_ESCAPE1 equ 216
```



```
                0000 00E0 OUTPUT_ESCAPE0 equ 224
                0000 01A2 PALLETTE equ 418
                0000 00E8 POLYGON equ 232
                0000 019C POLYGON_FILL_FACTOR equ 412
                0000 019E POLYGON_SOLID_FILL equ 414
                0000 0198 POLYGON_SUPPORT equ 408
                0000 019A REDEF_BACKGROUND equ 410
                0000 00F0 SET_MARKER equ 240
                0000 026A SINX_TABLE equ 618
                0000 027A SINY_TABLE equ 634
                0000 02D2 SOFT_CLIP_CPX equ 722
                0000 02D6 SOFT_CLIP_CPY equ 726
                0000 02C0 SOFT_CLIP_SAVEX0 equ 704
                0000 02C4 SOFT_CLIP_SAVEX1 equ 708
                0000 02C8 SOFT_CLIP_SAVEY0 equ 712
                0000 02CC SOFT_CLIP_SAVEY1 equ 716
                0000 02D0 SOFT_CLIP_SWITCH equ 720
                0000 02A4 SOFT_FONT_PTR equ 676
                0000 02A8 SOFT_TEXT_TEMP1 equ 680
                0000 02AC SOFT_TEXT_TEMP2 equ 684
                0000 015C SPOOLING equ 348
                0000 00F8 TEXT equ 248
                0000 01E2 TEXT_COS_DIR equ 482
                0000 0100 TEXT_DIR equ 256
                0000 0108 TEXT_JUST equ 264
                0000 0232 TEXT_LINE_X equ 562
                0000 0236 TEXT_LINE_Y equ 566
                0000 01DE TEXT_SIN_DIR equ 478
                0000 022A TEXT_SPACE_X equ 554
                0000 022E TEXT_SPACE_Y equ 558
                0000 0110 TEXT_SPACING equ 272
                0000 02B8 UNCLIPPED_DRAW equ 696
                0000 02B0 UNCLIPPED_MOVE equ 688
                0000 01A7 VECT_LINESTYLES equ 428


                          *
                          ********************************************************************************
                          *
                0000 0000 GLE_ASCLIP_GLE_SOFT_CLIP_MOVE equ *

00000000 4E56 0000              link    a6,#0
00000004 286E 0008              movea.l 8(a6),gcb               { A4 }

00000008 2F2C 01BA              move.l  end_x(gcb),-(sp)        { save ending point, this will }
0000000C 2F2C 01BE              move.l  end_y(gcb),-(sp)        { become the new cp            }

00000010 6100 0092              bsr clip                        { returns clipped points in }
                          *                                     { (d4,d5) and (d6,d7), d0 is}
                          *                                     { the clipped state opcode  }

00000014 B07C 0002              cmp.w   #2,d0                   {ck for total clip}
00000018 6714                   beq.s   move_complete

0000001A 2946 01BA              move.l  d6,end_x(gcb)           { pass clipped point }
0000001E 2947 01BE              move.l  d7,end_y(gcb)
```

```
00000022 2F0C            move.l    gcb,-(sp)                 { save copy of gcb on stack }
00000024 2F0C            move.l    gcb,-(sp)                 { call move routine }
00000026 206C 02B0       movea.l   unclipped_move(gcb),a0    { no static links }
0000002A 4E90            jsr       (a0)
0000002C 205F            movea.l   (sp)+,gcb                 { get gcb back off stack }

                  0000 002E move_complete equ *
0000002E 206F 01B6       move.l    (sp)+,current_pos_y(gcb)  { set cp to saved end_x, end_y }
00000032 206F 01B2       move.l    (sp)+,current_pos_x(gcb)  { saved on stack.  Note that  }
                         *                                   { the CP is saved in an unclipped }
                         *                                   { form                            }

00000036 4E5E            unlk      a6
00000038 2E9F            move.l    (sp)+,(sp)
0000003A 4E75            rts

                         ************************************************************************

                  0000 003C GLE_ASCLIP_GLE_SOFT_CLIP_DRAW equ *

0000003C 4E56 0000       link      a6,#0

00000040 286E 0008       movea.l   8(a6),gcb                 { A4 }
00000044 2F2C 01BA       move.l    end_x(gcb),-(sp)          { save ending point, this will }
00000048 2F2C 01BE       move.l    end_y(gcb),-(sp)          { become the new cp            }

0000004C 6100 0056       bsr clip                            { returns clipped points in }
                         *                                   { (d4,d5) and (d6,d7), d0 is}
                         *                                   { the clipped state opcode  }

00000050 4A40            tst.w     d0                        { ck for no clipping needed }
00000052 672E            beq.s     draw_it

00000054 B07C 0002       cmp.w     #2,d0                  {ck for line outside clipping bounds}
00000058 673C            beq.s     draw_complete

0000005A B86C 01B2       cmp.l     current_pos_x(gcb),d4     { ck to see if cp changed after }
0000005E 6606            bne.s     move_first                { clipping                      }
00000060 B8AC 01B6       cmp.l     current_pos_y(gcb),d5
00000064 671C            beq.s     draw_it

                  0000 0066 move_first equ *
00000066 2F06            move.l    d6,-(sp)                  { save end point on stack }
00000068 2F07            move.l    d7,-(sp)

0000006A 2D44 01BA       move.l    d4,end_x(gcb)             { pass clipped point }
0000006E 2D45 01BE       move.l    d5,end_y(gcb)

00000072 2F0C            move.l    gcb,-(sp)                 { save a copy of gcb pointer }
00000074 2F0C            move.l    gcb,-(sp)                 { call move routine }
00000076 206C 02B0       movea.l   unclipped_move(gcb),a0    { no static links }
0000007A 4E90            jsr       (a0)
0000007C 285F            movea.l   (sp)+,gcb                 { get gcb pointer back }

0000007E 2E1F            move.l    (sp)+,d7                  { pass clipped point }
00000080 2C1F            move.l    (sp)+,d6
```



```
                  0000 0082 draw_it equ *

00000082 2D46 01BA       move.l    d6,end_x(gcb)
00000086 2D47 01BE       move.l    d7,end_y(gcb)

0000008A 2F0C            move.l    gcb,-(sp)                 { save copy of gcb on stack }
0000008C 2F0C            move.l    gcb,-(sp)
0000008E 206C 02B8       movea.l   unclipped_draw(gcb),a0    { no static links }
00000092 4E90            jsr       (a0)
00000094 205F            movea.l   (sp)+,gcb                 { get gcb back }

                  0000 0096 draw_complete equ *

00000096 2F6F 01B6       move.l    (sp)+,current_pos_y(gcb)  { set cp to saved end_x, end_y }
0000009A 2F6F 01B2       move.l    (sp)+,current_pos_x(gcb)  { saved on stack.  Note that  }
                         *                                   { the CP is saved in an unclipped }
                         *                                   { form                            }

0000009E 4E5E            unlk      a6
000000A0 2E9F            move.l    (sp)+,(sp)
000000A2 4E75            rts
                         ************************************************************************
                         *
                         *
                         *  ASM clipping entry point is GLE_ASCLIP_CLIPPING
                         *
                         *  Regs = a0 - clip_xmin    d4 - x0
                         *         a1 - clip_ymin    d5 - y0
                         *         a2 - clip_xmax    d6 - x1
                         *         a3 - clip_ymax    d7 - y1
                         *
                         *         a4 - GLE_GCB
                         *
                         ************************************************************************
                         *
                         *       clip
                         *
                         *       d4=x0    a0=clip_xmin
                         *       d5=y0    a1=clip_ymin
                         *       d6=x1    a2=clip_xmax
                         *       d7=y1    a3=clip_ymax
                         *
                         *       Returns clipped points (x0,y0) and (x1,y1)
                         *       Returns opcode in d0 : 0 - not clipped
                         *                              1 - clipped but some part visible
                         *                              2 - all clipped
                         *
                  0000 00A4 clip    equ *

000000A4 4CEC 00F0       movem.l current_pos_x(gcb),d4-d7  { get x0,y0 x1,y1 }
         01B2
000000AA 4CEC 0F00       movem.l clip_limits_xmin(gcb),a0-a3
         01BE

000000B0 B8A8            cmp.l     clip_xmin,x0              fast check to find in bounds line
000000B2 6D00            blt.s     clip_it
```

```
000000B4 B889              cmp.l   clip_xmax,x0
000000B6 6E1C              bgt.s   clip_it
000000B8 BA8A              cmp.l   clip_ymin,y0
000000BA 6D18              blt.s   clip_it
000000BC BA8B              cmp.l   clip_ymax,y0
000000BE 6E14              bgt.s   clip_it

000000C0 BC88              cmp.l   clip_xmin,x1
000000C2 6D10              blt.s   clip_it
000000C4 BC89              cmp.l   clip_xmax,x1
000000C6 6E0C              bgt.s   clip_it
000000C8 BE8A              cmp.l   clip_ymin,y1
000000CA 6D08              blt.s   clip_it
000000CC BE8B              cmp.l   clip_ymax,y1
000000CE 6E04              bgt.s   clip_it

           0000 00D0 clipallin  equ *
000000D0 7000              moveq #0,d0             no clipping performed, set return opcode
000000D2 4E75              rts

           0000 00D4 GLE_ASCLIP_CLIPPING equ *
000000D4 2F0D      clip_it move.l  a5,-(sp)        (save global base, and free up a5)
000000D6 426C 02D0         clr.w   soft_clip_switch(gcb)
000000DA BA87              cmp.l   y1,y0           check for horizontal lines
000000DC 6700 015C         beq     clip_horizontal

000000E0 6D0A              blt.s   clip1           force y0 < y1
000000E2 C946              exg     x0,x1
000000E4 CB47              exg     y0,y1
000000E6 086C 0000         bchg    #0,soft_clip_switch(gcb)
           02D0
000000EC BA8B      clip1   cmp.l   clip_ymax,y0    ck for both y above
000000EE 6E00 0144         bgt     clip_out
000000F2 BE8A              cmp.l   clip_ymin,y1    ck for both y below
000000F4 6D00 013E         blt     clip_out

000000F8 B886              cmp.l   x1,x0           check for vertical lines
000000FA 6700 016C         beq     clip_vertical

000000FE 6D0A              blt.s   clip2           now force x0 < x1
00000100 C946              exg     x0,x1
00000102 CB47              exg     y0,y1
00000104 086C 0000         bchg    #0,soft_clip_switch(gcb)
           02D0
0000010A B889      clip2   cmp.l   clip_xmax,x0    ck for both x left
0000010C 6E00 0126         bgt     clip_out
00000110 BC88              cmp.l   clip_xmin,x1    ck for both x right
00000112 6D00 0120         blt     clip_out

                   *
                   * At this point, a diagonal line exists with one or two ends
                   * outside the current clipping limits.  This line is reduced by replacing
                   * its end points with window intersections.  Intersections are found
                   * by using the midpoint clipping procedure (Newman&Sproull)
                   *
```



```
00000116 2944 02C0 solv_x1 move.l  x0,soft_clip_savex0(gcb)  save original points
0000011A 2945 02C8         move.l  y0,soft_clip_savey0(gcb)
0000011E 2946 02C4         move.l  x1,soft_clip_savex1(gcb)
00000122 2947 02CC         move.l  y1,soft_clip_savey1(gcb)

00000126 B888              cmp.l   clip_xmin,x0    is x0 inside?
00000128 6C1E              bge.s   solv_xr

0000012A 2A48      solv_x1 movea.l  clip_xmin,a5    solve for y at (x = xmin)
0000012C 2004              move.l  x0,d0           pass parms to solve
0000012E 2205              move.l  y0,d1           (note: x0 = d4, y0 = d5)
00000130 2605              move.l  y0,d3   (lefty)
00000132 2404              move.l  x0,d2   (leftx)
00000134 2A07              move.l  y1,d5   (righty)
00000136 2806              move.l  x1,d4   (rightx)
00000138 6100 014E         bsr     solve
0000013C 2800              move.l  d0,x0
0000013E 2A01              move.l  d1,y0

00000140 2944 02C0         move.l  x0,soft_clip_savex0(gcb)
00000144 2945 02C8         move.l  y0,soft_clip_savey0(gcb)

00000148 BC89      solv_xr cmp.l   clip_xmax,x1    is x1 inside?
0000014A 6F2E              ble.s   intsct3

0000014C 2A49              movea.l  clip_xmax,a5    solve for y at (x = xmax)
0000014E 2006              move.l  x1,d0           pass parms to solve
00000150 2207              move.l  y1,d1           (note: x0 = d4, y0 = d5)
00000152 2C04              move.l  x0,x1           free up d4, d5 (x1, an y1 are now free)
00000154 2E05              move.l  y0,y1
00000156 2A2C 02CC         move.l  soft_clip_savey1(gcb),d5
0000015A 282C 02C4         move.l  soft_clip_savex1(gcb),d4
0000015E 262C 02C8         move.l  soft_clip_savey0(gcb),d3
00000162 242C 02C0         move.l  soft_clip_savex0(gcb),d2
00000166 6100 0120         bsr     solve
0000016A 2806              move.l  x1,x0           restore x0,y0
0000016C 2A07              move.l  y1,y0
0000016E 2C00              move.l  d0,x1
00000170 2E01              move.l  d1,y1

00000172 2946 02C4         move.l x1,soft_clip_savex1(gcb)
00000176 2947 02CC         move.l y1,soft_clip_savey1(gcb)

0000017A BA87      intsct3 cmp.l   y1,y0           force y2 > y1
0000017C 6D26              blt.s   intsct4

0000017E C946              exg     x0,x1
00000180 CB47              exg     y0,y1
00000182 202C 02C0         move.l  soft_clip_savex0(gcb),d0        swap saved copies as well
00000186 296C 02C4         move.l  soft_clip_savex1(gcb),soft_clip_savex0(gcb)
           02C0
0000018C 2940 02C4         move.l  d0,soft_clip_savex1(gcb)
00000190 202C 02C8         move.l  soft_clip_savey0(gcb),d0
00000194 296C 02CC         move.l  soft_clip_savey1(gcb),soft_clip_savey0(gcb)
           02C8
```

```
 452  0000019A 2943 02CC          move.l  d0,soft_clip_savey1(gcb)
 453  0000019E 0868 0000          bchg    #0,soft_clip_switch(gcb)
                02D0
 454
 455  000001A4 BA8B       intsct4 cmp.l   clip_ymax,y0
 456  000001A6 6E00 008C          bgt     clip_out
 457  000001AA BE8A               cmp.l   clip_ymin,y1
 458  000001AC 6D00 0086          blt     clip_out
 459
 460  000001B0 BA8A       solv_yu cmp.l   clip_ymin,y0
 461  000001B2 6C2A               bge.s   solv_yd
 462
 463  000001B4 2A4A               movea.l  clip_ymin,a5          solve for x at (y = ymin)
 464  000001B6 2628 02C0          move.l  soft_clip_savex0(gcb),d3      Pass parms to solve
 465  000001BA 2428 02C8          move.l  soft_clip_savey0(gcb),d2
 466  000001BE 2028 02C8          move.l  soft_clip_savey0(gcb),d0
 467  000001C2 2228 02C0          move.l  soft_clip_savex0(gcb),d1
 468  000001C6 2A28 02C4          move.l  soft_clip_savex1(gcb),d5
 469  000001CA 2828 02CC          move.l  soft_clip_savey1(gcb),d4
 470  000001CE 6100 00B8          bsr     solve
 471  000001D2 2A00               move.l  d0,y0
 472  000001D4 2801               move.l  d1,x0
 473
 474  000001D6 2945 02C8          move.l  y0,soft_clip_savey0(gcb)
 475  000001DA 2944 02C0          move.l  x0,soft_clip_savex0(gcb)
 476
 477  000001DE BE8B       solv_yd cmp.l   clip_ymax,y1
 478  000001E0 6F00 002C          ble     clip_in
 479
 480  000001E4 2A4B               movea.l clip_ymax,a5          solve for x at (y = ymin)
 481  000001E6 2C04               move.l  x0,x1                 (save x0,y0 in x1,y1 which is free)
 482  000001E8 2E05               move.l  y0,y1
 483  000001EA 2A28 02C4          move.l  soft_clip_savex1(gcb),d5
 484  000001EE 2828 02CC          move.l  soft_clip_savey1(gcb),d4
 485  000001F2 2628 02C0          move.l  soft_clip_savex0(gcb),d3
 486  000001F6 2428 02C8          move.l  soft_clip_savey0(gcb),d2
 487  000001FA 2028 02CC          move.l  soft_clip_savey1(gcb),d0
 488  000001FE 2228 02C4          move.l  soft_clip_savex1(gcb),d1
 489  00000202 6100 0084          bsr     solve
 490  00000206 2806               move.l  x1,x0                 restore x0,y0
 491  00000208 2A07               move.l  y1,y0
 492  0000020A 2E00               move.l  d0,y1
 493  0000020C 2C01               move.l  d1,x1
 494
 495            0000 020E clip_in equ *
 496  0000020E 0828 0000          btst    #0,soft_clip_switch(gcb)
                02D0
 497  00000214 6704               beq.s   clipin
 498  00000216 C946               exg     x0,x1             restore org direction
 499  00000218 CB47               exg     y0,y1
 500            0000 021A clipin  equ *
 501  0000021A 7001               moveq   #1,d0               clipping one or more points
 502  0000021C 2A5F               movea.l (sp)+,a5                  (restore global base)
 503  0000021E 4E75               rts
 504
 505            0000 0220 clip_all_in equ *
 506  00000220 2A5F               movea.l (sp)+,a5                  (restore global base)
```



```
 507  00000222 0828 0000          btst    #0,soft_clip_switch(gcb)
                02D0
 508  00000228 6700 FEA6          beq     clipallin
 509  0000022C C946               exg     x0,x1             restore org direction
 510  0000022E CB47               exg     y0,y1
 511  00000230 6000 FE9E          bra     clipallin
 512
 513            0000 0234 clip_out equ *
 514  00000234 7002               moveq   #2,d0               all clipped, set up return opcode
 515  00000236 2A5F               movea.l (sp)+,a5                  (restore global base)
 516  00000238 4E75               rts
 517
 518            0000 023A clip_horizontal equ *
 519  0000023A BE8B               cmp.l   clip_ymax,y1      ck for y out of clip limits
 520  0000023C 6EF6               bgt     clip_out
 521  0000023E BE8A               cmp.l   clip_ymin,y1
 522  00000240 6DF2               blt     clip_out
 523
 524  00000242 B88B               cmp.l   x1,x0
 525  00000244 6700 0038          beq     clip_dot
 526  00000248 6D08               blt.s   clip_h1
 527  0000024A C946               exg     x0,x1
 528  0000024C 0868 0000          bchg    #0,soft_clip_switch(gcb)
                02D0
 529  00000252 B889       clip_h1 cmp.l   clip_xmax,x0      x0 < x1
 530  00000254 6EDE               bgt     clip_out
 531  00000256 BC88               cmp.l   clip_xmin,x1
 532  00000258 6DDA               blt     clip_out
 533  0000025A BC89               cmp.l   clip_xmax,x1
 534  0000025C 6F02               ble.s   clip_h2
 535  0000025E 2C09               move.l  clip_xmax,x1
 536  00000260 B888       clip_h2 cmp.l   clip_xmin,x0
 537  00000262 6CAA               bge.s   clip_in
 538  00000264 2808               move.l  clip_xmin,x0
 539  00000266 60A6               bra.s   clip_in
 540
 541            0000 0268 clip_vertical equ *
 542
 543  00000268 BC89               cmp.l   clip_xmax,x1
 544  0000026A 6EC8               bgt     clip_out
 545  0000026C BC88               cmp.l   clip_xmin,x1
 546  0000026E 6DC4               blt     clip_out
 547  00000270 BE8B               cmp.l   clip_ymax,y1          assumes y0 < y1
 548  00000272 6F02               ble.s   clip_v1
 549  00000274 2E0B               move.l  clip_ymax,y1
 550  00000276 BA8A       clip_v1 cmp.l   clip_ymin,y0
 551  00000278 6C94               bge     clip_in
 552  0000027A 2A0A               move.l  clip_ymin,y0
 553  0000027C 6090               bra     clip_in
 554
 555            0000 027E clip_dot      equ *
 556  0000027E B889               cmp.l   clip_xmax,x0
 557  00000280 6EB2               bgt     clip_out
 558  00000282 B888               cmp.l   clip_xmin,x0
 559  00000284 6DAE               blt     clip_out
 560  00000286 6088               bra     clip_in
 561
```

```
                        * routine to find intersections.  D0 will be driven to the value supplied
                        * as val_x.  D1 will contain the corresponding value.  "left" and "right"
                        * contain the original starting points.  "X" and "Y" are only reletive
                        * to the current use of this routine, not the physical bounds of the
                        * display
                        *
              0000 0288 solve   equ *

              0000 0005 val_x   equ a5
              0000 0005 right_y equ d5
              0000 0004 right_x equ d4
              0000 0003 left_y  equ d3
              0000 0002 left_x  equ d2

     00000288 B08D      solve_s cmp.l   val_x,d0
     0000028A 672C              beq.s   solve_e
     0000028C 6D0E              blt.s   solve_l

     0000028E 2800      solve_r move.l  d0,right_x
     00000290 2A01              move.l  d1,right_y
     00000292 D082              add.l   left_x,d0
     00000294 6920              bvs.s   force_exit        check for overflow
     00000296 D283              add.l   left_y,d1
     00000298 691C              bvs.s   force_exit        check for overflow
     0000029A 6014              bra.s   round

     0000029C 2400      solve_l move.l  d0,left_x
     0000029E 2601              move.l  d1,left_y
     000002A0 D084              add.l   right_x,d0
     000002A2 6912              bvs.s   force_exit        check for overflow
     000002A4 5280              addq.l  #1,d0
     000002A6 690E              bvs.s   force_exit        check for overflow
     000002A8 D285              add.l   right_y,d1
     000002AA 690A              bvs.s   force_exit        check for overflow
     000002AC 5281              addq.l  #1,d1
     000002AE 6906              bvs.s   force_exit        check for overflow

     000002B0 E280      round   asr.l   #1,d0
     000002B2 E281              asr.l   #1,d1
     000002B4 60D2              bra.s   solve_s

     000002B6 4E76      force_exit trapv                  let system process

     000002B8 4E75      solve_e rts

     000002BA 4E75      gle_asclip_gle_asclip  rts

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

ASM_STEXT

Description

ASM_STEXT provides software graphics text routines.

Usage

Called as a DGL "tool."

```
PASS 1 COMPLETE. ERRORS: 0
    1                       *
    2                       *  Graphics Low End
    3                       *
    4                       *  Module     = Software text
    5                       *  Programer  = BJS
    6                       *  Date       = 9-30-82
    7                       *
    8                       *  Purpose : To provide software text generation routines.
    9                       *
   10                       *  Rev history
   11                       *
   12                       *    Created    - 9-30-82
   13                       *    Modified   - 6-27-83  BJS  Added export text for soft_text routine
   14                       *
   15                       *
   16                       *     (c) Copyright Hewlett-Packard Company, 1983.
   17                       *     All rights are reserved.  Copying or other
   18                       *     reproduction of this program except for archival
   19                       *     purposes is prohibited without the prior
   20                       *     written consent of Hewlett-Packard Company.
   21                       *
   22                       *
   23                       *                 RESTRICTED RIGHTS LEGEND
   24                       *
   25                       *     Use, duplication, or disclosure by the Government
   26                       *     is subject to restrictions as set forth in
   27                       *     paragraph (b) (3) (B) of the Rights in Technical
   28                       *     Data and Computer Software clause in
   29                       *     DAR 7-104.9(a).
   30                       *
   31                       *     HEWLETT-PACKARD COMPANY
   32                       *     Fort Collins, Colorado
   33                       *
   34                       *
   35                            mname GLE_ASTEXT
   36
   37                              src module GLE_ASTEXT;
   38                              src import GLE_TYPES;
   39                              src export
   40                              src procedure gle_soft_text ( gcb : graphics_control_block_ptr );
   41                              src end;
   42
   43                              nosyms
   44                       *
   45                       *  Export text is defined by STEXT Pascal module
   46                       *
   47                       *
   48                       *  Define entry points
   49                       *
   50  00000000                  rorg 0
   51                            def GLE_ASTEXT_GLE_SOFT_TEXT
   52                            def GLE_ASTEXT_GLE_ASTEXT
   53                       *
   54                       *    Set up globals
   55                       *
   56            0000 0000 GCB equ a0
   57
   58                            INCLUDE ASM_TYPES
```



```
   59            0000 0018 AWAIT_BLANKING equ 24
   60            0000 018E BACKGROUND equ 398
   61            0000 0020 BUFFER_MODE equ 32
   62            0000 029C CALC_SOFT_TEXT_XFORM equ 668
   63            0000 029A CALC_TEXT_XFORM equ 666
   64            0000 01D2 CHAR_HEIGHT equ 466
   65            0000 01E6 CHAR_JUST_X equ 486
   66            0000 01EA CHAR_JUST_Y equ 490
   67            0000 0028 CHAR_SIZE equ 40
   68            0000 01B0 CHAR_SIZES equ 432
   69            0000 01D6 CHAR_SPACE equ 470
   70            0000 01CE CHAR_WIDTH equ 462
   71            0000 0030 CLEAR equ 48
   72            0000 0038 CLIP_LIMITS equ 56
   73            0000 01F2 CLIP_LIMITS_XMAX equ 498
   74            0000 01EE CLIP_LIMITS_XMIN equ 494
   75            0000 01FA CLIP_LIMITS_YMAX equ 506
   76            0000 01F6 CLIP_LIMITS_YMIN equ 502
   77            0000 0196 COLOR_MAP_SUPPORT equ 406
   78            0000 0190 COMPLEMENT_SUPPORT equ 400
   79            0000 01AA CONT_LINESTYLES equ 426
   80            0000 023A COSX_TABLE equ 570
   81            0000 024A COSY_TABLE equ 586
   82            0000 0208 CURRENT_BUFFER_MODE equ 520
   83            0000 0212 CURRENT_COLOR_INDEX equ 530
   84            0000 01FE CURRENT_CURSOR_STATE equ 510
   85            0000 0200 CURRENT_CURSOR_X equ 512
   86            0000 0204 CURRENT_CURSOR_Y equ 516
   87            0000 0216 CURRENT_DRAWING_MODE equ 534
   88            0000 0214 CURRENT_FILL_INDEX equ 532
   89            0000 020A CURRENT_LINESTYLE equ 522
   90            0000 0210 CURRENT_LINESTYLE_MODE equ 528
   91            0000 020C CURRENT_LINESTYLE_PATTERN equ 524
   92            0000 0218 CURRENT_LINEWIDTH equ 536
   93            0000 020E CURRENT_PATTERN_LENGTH equ 526
   94            0000 0220 CURRENT_POLYGON_BLUE equ 544
   95            0000 021A CURRENT_POLYGON_COLOR equ 538
   96            0000 021E CURRENT_POLYGON_GREEN equ 542
   97            0000 021C CURRENT_POLYGON_RED equ 540
   98            0000 01B2 CURRENT_POS_X equ 434
   99            0000 01B6 CURRENT_POS_Y equ 438
  100            0000 0040 CURSOR equ 64
  101            0000 0048 DEFINE_COLOR_MAP equ 72
  102            0000 0050 DEFINE_DRAWING_MODE equ 80
  103            0000 014C DEVICE_BUF equ 332
  104            0000 0154 DEVICE_INFO equ 340
  105            0000 0158 DEVICE_INFO_CHAR_COUNT equ 344
  106            0000 0150 DEV_DEP_STUFF equ 336
  107            0000 016C DISPLAY_HANDLER_CHAR_COUNT equ 364
  108            0000 0166 DISPLAY_HANDLER_NAME equ 358
  109            0000 0186 DISPLAY_MAX_X equ 390
  110            0000 018A DISPLAY_MAX_Y equ 394
  111            0000 017E DISPLAY_MIN_X equ 382
  112            0000 0182 DISPLAY_MIN_Y equ 386
  113            0000 015E DISPLAY_NAME equ 350
  114            0000 0164 DISPLAY_NAME_CHAR_COUNT equ 356
  115            0000 016E DISPLAY_RES_X equ 366
```

```
                0000 0176 DISPLAY_RES_Y equ 374
                0000 01A0 DITHER_SUPPORT equ 416
                0000 0058 DRAW equ 88
                0000 0118 DUMMY_XXX equ 280
                0000 01BA END_X equ 442
                0000 01BE END_Y equ 446
                0000 0194 ERASE_SUPPORT equ 404
                0000 015A ERROR_RETURN equ 346
                0000 0060 FILL_INDEX_COLOR equ 96
                0000 0068 FLUSH_BUFFER equ 104
                0000 01A6 GAMUT equ 422
                0000 0070 GET_COLOR_MAP equ 112
                0000 0080 GET_POLYGON_INFO equ 128
                0000 0078 GET_RASTER equ 120
                0000 0088 GLOAD equ 136
                0000 0090 GRAPHICS_ON_OFF equ 144
                0000 0098 GSTORE equ 152
                0000 00A0 INDEX_COLOR equ 160
                0000 0000 INFO1 equ 0
                0000 0004 INFO2 equ 4
                0000 0008 INFO3 equ 8
                0000 000C INFO4 equ 12
                0000 0010 INFO_PTR1 equ 16
                0000 0014 INFO_PTR2 equ 20
                0000 00A8 INQ_P1P2 equ 168
                0000 0148 IOCB equ 328
                0000 0120 IO_INQ_TIMEOUT equ 288
                0000 0128 IO_REFD equ 296
                0000 0130 IO_SET_TIMEOUT equ 304
                0000 0138 IO_TERM equ 312
                0000 0140 IO_WRITE equ 320
                0000 01CC KATA equ 460
                0000 00B8 LINESTYLE equ 184
                0000 00B0 LINEWIDTH equ 176
                0000 01AE LINEWIDTHS equ 430
                0000 01DA LINE_SPACE equ 474
                0000 00C0 MARKER equ 192
                0000 01C8 MARKER_HEIGHT equ 456
                0000 00C8 MARKER_SIZE equ 200
                0000 01C2 MARKER_TYPE equ 450
                0000 01C4 MARKER_WIDTH equ 452
                0000 00D0 MOVE equ 208
                0000 0192 NON_DOMINANT_SUPPORT equ 402
                0000 0222 OLD_A5 equ 546
                0000 0226 OLD_A6 equ 550
                0000 00D8 OUTPUT_ESCAPE1 equ 216
                0000 00E0 OUTPUT_ESCAPE0 equ 224
                0000 01A2 PALLETTE equ 418
                0000 00E8 POLYGON equ 232
                0000 019C POLYGON_FILL_FACTOR equ 412
                0000 019E POLYGON_SOLID_FILL equ 414
                0000 0198 POLYGON_SUPPORT equ 408
                0000 019A REDEF_BACKGROUND equ 410
                0000 00F0 SET_MARKER equ 240
                0000 028A SINX_TABLE equ 618
                0000 027A SINY_TABLE equ 634
                0000 02D2 SOFT_CLIP_CPX equ 722
```



```
                0000 02D6 SOFT_CLIP_CPY equ 726
                0000 02C0 SOFT_CLIP_SAVEX0 equ 704
                0000 02C4 SOFT_CLIP_SAVEX1 equ 708
                0000 02C8 SOFT_CLIP_SAVEY0 equ 712
                0000 02CC SOFT_CLIP_SAVEY1 equ 716
                0000 02D0 SOFT_CLIP_SWITCH equ 720
                0000 02A4 SOFT_FONT_PTR equ 676
                0000 02A8 SOFT_TEXT_TEMP1 equ 680
                0000 02AC SOFT_TEXT_TEMP2 equ 684
                0000 015C SPOOLING equ 348
                0000 00F8 TEXT equ 248
                0000 01E2 TEXT_COS_DIR equ 482
                0000 0100 TEXT_DIR equ 256
                0000 0108 TEXT_JUST equ 264
                0000 0232 TEXT_LINE_X equ 562
                0000 0236 TEXT_LINE_Y equ 566
                0000 01DE TEXT_SIN_DIR equ 478
                0000 022A TEXT_SPACE_X equ 554
                0000 022E TEXT_SPACE_Y equ 558
                0000 0110 TEXT_SPACING equ 272
                0000 02B8 UNCLIPPED_DRAW equ 696
                0000 02B0 UNCLIPPED_MOVE equ 688
                0000 01AC VECT_LINESTYLES equ 428


                0000 0000 cosx_tab equ 0
                0000 0010 cosy_tab equ 16
                0000 0030 sinx_tab equ 48
                0000 0040 siny_tab equ 64
                          *
                          ********************************************************************************
                          *
                0000 0000 GLE_ASTEXT_GLE_SOFT_TEXT equ *

00000000 4E41             trap #1             stack overflow ck
00000002 0000             dc.w 0
                          *
                          * Check calc_text_xform to see if the transformation needs
                          * to be re-calculated.
                          *
00000004 206E 0008        movea.l  8(a6),gcb                        (A0)
00000008 4A68 029A        tst.w    calc_text_xform(gcb)
0000000C 6700 000A        beq      xform_ok

00000010 2F08             move.l   gcb,-(sp)                        (pass gcb)
00000012 2068 029C s0     movea.l  calc_soft_text_xform(gcb),a0     ( no static links )
00000016 4E90             jsr      (a0)

                0000 0018 xform_ok equ *
                          *
                          * Get GCB pointer
                          * Move string address to local address reg
                          * Move cnt to local address reg
                          * Get addr of stroke table in local address reg
                          *
00000018 206E 0008        movea.l  8(a6),gcb                  (A0)
```

```
PAGE    5 [3.0] 12/28/84 22:28:37  ASSEMBLY OF ASM_STEXT.TEXT                                   *** File name: ASM_STEXT ***

  230    0000001C 2268 0010       movea.l  info_ptr1(gcb),a1          {string ptr}
  231    00000020 2028 0000       move.l   info1(gcb),d0              {string count}
  232                             *
  233                             *
  234                             *     save a copy of starting cp in temp (this is used for CR point)
  235                             *
  236    00000024 2228 01B2       move.l   current_pos_x(gcb),d1
  237    00000028 2E28 01B6       move.l   current_pos_y(gcb),d7
  238    0000002C 2141 02A8       move.l   d1,soft_text_temp1(gcb)
  239    00000030 2147 02AC       move.l   d7,soft_text_temp2(gcb)
  240    00000034 247C 0000       movea.l  #0,a4
                  0000
  241                             *
  242              0000 003A NEXT_CHAR equ *
  243              0000 003A bad_char equ *             {bad characters are ignored}
  244                             *
  245                             * Top of loop
  246                             * Check character count; Quit if 0
  247                             *
  248    0000003A 907C 0001       sub.w    #1,d0              {=0?}
  249    0000003E 6D00 01A4       blt      DONE
  250                             *
  251                             * Get current character from string
  252                             *
  253    00000042 7200            moveq    #0,d1              {clear high byte}
  254    00000044 1219            move.b   (a1)+,d1           {d1 <- char}
  255    00000046 D8FC 0000       adda.l   #1,a4
                  0001
  256                             *
  257                             * Calc which font tables to use
  258                             *
  259    0000004C 2468 02A4       movea.l  soft_font_ptr(gcb),A2    {Pointer to stroke tables}
  260    00000050 2E4A            movea.l  a2,a3              {copy stroke table pointer}
  261
  262    00000052 907C 0020       sub.w    #32,d1             {all char < 32 are control}
  263    00000056 6D00 011C       blt      control
  264
  265    0000005A B27C 005F       cmp.w    #95,d1             {ck for char > 127}
  266    0000005E 6E00 0016       bgt      plot_hi
  267
  268    00000062 B27C 003C       cmp.w    #60,d1             {ck for sqr root (special kata)
  269    00000066 6600 0044       bne      plot_std
  270
  271    0000006A 4A68 01CC       tst.w    kata(gcb)          {ck katakana flag}
  272    0000006E 673C            beq.s    plot_std
  273    00000070 723F            moveq    #63,d1             {use kata char #63 for sqr root}
  274    00000072 6000 0014       bra      plot_kata
  275
  276              0000 0076 plot_hi equ *
  277    00000076 4A68 01CC       tst.w    kata(gcb)          {ck katakana flag}
  278    0000007A 6718            beq.s    plot_rom
  279    0000007C 927C 0081       sub.w    #129,d1            {legal kata from 161 to 223}
  280    00000080 6CB8            blt      bad_char
  281    00000082 B27C 003E       cmp.w    #62,d1
  282    00000086 6FB2            bgt      bad_char
  283
  284              0000 0088 plot_kata equ *
```

```
PAGE    6 [3.0] 12/28/84 22:28:37  ASSEMBLY OF ASM_STEXT.TEXT                                   *** File name: ASM_STEXT ***

  285                             *
  286    00000088 D7EA 0014       adda.l   $14(a2),a3         {A3 points to top of KATA Pointer table}
  287    0000008C D5EA 0010       adda.l   $10(a2),a2         {A2 points to top of KATA stroke table}
  288    00000090 6000 0020       bra      plot_common
  289
  290              0000 0094 plot_rom  equ *
  291    00000094 927C 0088       sub.w    #136,d1            {legal roman are 168 to 222}
  292    00000098 6DA0            blt      bad_char
  293    0000009A B27C 0036       cmp.w    #54,d1
  294    0000009E 6E9A            bgt      bad_char
  295
  296                             *
  297    000000A0 D7EA 000C       adda.l   $C(a2),a3          {A3 points to top of KATA Pointer table}
  298    000000A4 D5EA 0008       adda.l   $8(a2),a2          {A2 points to top of KATA stroke table}
  299    000000A8 6000 0008       bra      plot_common
  300
  301              0000 00AC plot_std  equ *                {STD is first entry in stroke table}
  302                             *
  303    000000AC D7EA 0004       adda.l   4(a2),a3           {A3 points to top of Pointer table}
  304    000000B0 D5D2            adda.l   (a2),a2            {A2 points to top of stroke table}
  305                             *
  306                             ********************************************************************************
  307                             *
  308              0000 00B2 plot_common equ *
  309                             *
  310                             * Calc number of vectors in character by indexing into pointer sub-table
  311                             *
  312    000000B2 D241            add.w    d1,d1              {calc 16 bit offset with character}
  313    000000B4 3A33 1002       move.w   2(a3,d1),d5        {get index of first vector of next char}
  314    000000B8 3C33 1000       move.w   0(a3,d1),d6        {get index of first vector of this char}
  315    000000BC 9A46            sub.w    d6,d5              {d5 is the number of vectors in char}
  316                             *
  317                             * Calc indexs to vector locations in stroke table
  318                             *
  319    000000BE 967C 0001       sub.w    #1,d6              {pointer are base 1, we need base 0}
  320    000000C2 D4C6            adda.w   d6,a2              {form ptr to first vector}
  321                             *
  322                             *     move current position into local regs
  323                             *
  324    000000C4 2228 01B2       move.l current_pos_x(gcb),d1  {X}
  325    000000C8 2E28 01B6       move.l current_pos_y(gcb),d7  {Y}
  326                             *
  327              0000 00CC VECTOR_LOOP equ *
  328                             *
  329                             * Top of vector loop
  330                             *
  331    000000CC 181A            move.b   (a2)+,d4           { get packed vector information }
  332
  333    000000CE 1404            move.b   d4,d2              { unpack x information }
  334    000000D0 E84A            lsr.w    #4,d2
  335    000000D2 C47C 0007       and.w    #$0007,d2
  336    000000D6 D442            add.w    d2,d2              { form word index into cos table }
  337
  338    000000D8 1604            move.b   d4,d3              { unpack y information }
  339    000000DA C67C 000F       and.w    #$000f,d3
  340    000000DE D643            add.w    d3,d3              { form word index into cos table }
  341
```

```
PAGE   7 [3.0] 12/26/84 22:28:37  ASSEMBLY OF ASM_STEXT.TEXT                                  *** File name: ASM_STEXT ***

  342    000000E0 2648             movea.l  gcb,a3
  343    000000E2 D7FC 0000        adda.l   #cosx_table,a3
                  023A
  344    000000E8 3C33 2000        move.w   cosx_tab(a3,d2.w),d6
  345    000000EC DC73 3040        add.w    siny_tab(a3,d3.w),d6  ( d6 is new X )
  346    000000F0 48C6             ext.l    d6
  347
  348    000000F2 3633 3010        move.w   cosy_tab(a3,d3.w),d3
  349    000000F6 D673 2030        add.w    sinx_tab(a3,d2.w),d3  ( d3 is new Y )
  350    000000FA 48C3             ext.l    d3
  351
  352    000000FC DC81             add.l    d1,d6          {translate by CP}
  353    000000FE 6900 00EA        bvs      range_error
  354    00000102 D687             add.l    d7,d3
  355    00000104 6900 00E4        bvs      range_error
  356
  357    00000108 48E7 C5E8        movem.l  d0/d1/d5/d7/a0-a2/a4,-(sp) {save local state}
  358
  359    0000010C 2F08             move.l   gcb,-(sp)      {pass gcb}
  360
  361    0000010E 2146 01BA        move.l   d6,end_x(gcb)  {pass parms to vector generator}
  362    00000112 2143 01BE        move.l   d3,end_y(gcb)
  363
  364    00000116 0804 0007        btst     #7,d4          {ck control for move or draw}
  365    0000011A 670A             beq.s    needmove
  366
  367    0000011C 2068 0058        movea.l  draw(gcb),a0   ( no static links )
  368    00000120 4E90             jsr      (a0)
  369    00000122 6000 0008        bra  next_vector
  370
                  0000 0126 NEEDMOVE equ *
  372
  373    00000126 2068 00D0        movea.l  move(gcb),a0   ( no static links )
  374    0000012A 4E90             jsr      (a0)
  375
                  0000 012C NEXT_VECTOR equ *
  377    0000012C 4CDF 17A3        movem.l  (sp)+,d0/d1/d5/d7/a0-a2/a4 {restore local state}
  378
                  0000 0130 TEST_VECTOR equ *
  380                       *
  381                       * Check for no vectors left
  382                       *
  383    00000130 51CD FF9A        dbra     d5,VECTOR_LOOP
  384                       *
  385                       * Update CP
  386                       *
                  0000 0134 update_cp equ *
  388    00000134 4282             clr.l    d2
  389    00000136 340C             move     a4,d2
  390    00000138 2228 022A        move.l   text_space_x(gcb),d1
  391    0000013C 3641             movea.w  d1,a3
  392    0000013E B7C1             cmpa.l   d1,a3
  393    00000140 6600 00A8        bne      range_error
  394    00000144 2E28 022E        move.l   text_space_y(gcb),d7
  395    00000148 3647             movea.w  d7,a3
  396    0000014A B7C7             cmpa.l   d7,a3
  397    0000014C 6600 009C        bne      range_error
```

```
PAGE   8 [3.0] 12/26/84 22:28:37  ASSEMBLY OF ASM_STEXT.TEXT                                  *** File name: ASM_STEXT ***

  398    00000150 C3C2             muls     d2,d1
  399    00000152 CFC2             muls     d2,d7
  400    00000154 E681             asr.l    #3,d1
  401    00000156 E687             asr.l    #3,d7
  402    00000158 D2A8 02A8        add.l    soft_text_temp1(gcb),d1
  403    0000015C 6900 008C        bvs      range_error
  404    00000160 DEA8 02AC        add.l    soft_text_temp2(gcb),d7
  405    00000164 6900 0084        bvs      range_error
  406
                  0000 0168 UPDATE equ *
  408
  409    00000168 2141 01B2        move.l   d1,current_pos_x(gcb)
  410    0000016C 2147 01B6        move.l   d7,current_pos_y(gcb)
  411    00000170 6000 FEC8        bra      NEXT_CHAR
  412                       *
                  0000 0174 control  equ *
  414
  415    00000174 D27C 0020        add.w    #32,d1         {restore character value}
  416    00000178 B27C 000D        cmp.w    #13,d1         ( CR ? )
  417    0000017C 6716             beq.s    C_return
  418
  419    0000017E B27C 000A        cmp.w    #10,d1         ( LF ? )
  420    00000182 6728             beq.s    L_feed
  421
  422    00000184 B27C 0008        cmp.w    #8,d1          ( BS ? )
  423    00000188 6600 FEB0        bne      bad_char
  424
                  0000 018C B_space equ *
  426
  427    0000018C 99FC 0000        suba.l   #2,a4
                  0002
  428    00000192 60A0             bra.s    update_cp
  429
                  0000 0194 C_return equ *
  431
  432    00000194 287C 0000        movea.l  #0,a4
                  0000
  433    0000019A 2228 01B2        move.l   current_pos_x(gcb),d1
  434    0000019E 2E28 01B6        move.l   current_pos_y(gcb),d7
  435    000001A2 2228 02A8        move.l   soft_text_temp1(gcb),d1    ( restore cp to begining )
  436    000001A6 2E28 02AC        move.l   soft_text_temp2(gcb),d7
  437    000001AA 60BC             bra      UPDATE                 ( Update the CP and process next char )
  438
                  0000 01AC L_feed equ *
  440
  441                       *
  442                       * Update local cp (d1,c7)
  443                       * Update starting cp (temp1, temp2)
  444                       *
  445    000001AC 99FC 0000        suba.l    #1,a4
                  0001
  446    000001B2 2228 01B2        move.l   current_pos_x(gcb),d1
  447    000001B6 2E28 01B6        move.l   current_pos_y(gcb),d7
  448
  449    000001BA 2428 0232        move.l   text_line_x(gcb),d2
  450    000001BE 2628 0236        move.l   text_line_y(gcb),d3
  451
```

```
PAGE   9 [3.0] 12/26/84 22:28:37  ASSEMBLY OF ASM_STEXT.TEXT                                  *** File name: ASM_STEXT ***

  452    000001C2 E682              asr.l     #3,d2
  453    000001C4 E683              asr.l     #3,d3
  454
  455    000001C6 D282              add.l     d2,d1                 { inc local cp }
  456    000001C8 6900 0020         bvs       range_error
  457    000001CC DE83              add.l     d3,d7
  458    000001CE 6900 001A         bvs       range_error
  459
  460    000001D2 D5A8 02A8         add.l     d2,soft_text_temp1(gcb)
  461    000001D6 6900 0012         bvs       range_error
  462    000001DA D7A8 02AC         add.l     d3,soft_text_temp2(gcb)
  463    000001DE 6900 000A         bvs       range_error
  464
  465    000001E2 6084              bra       UPDATE                { Update the CP and process next char }
  466
  467              0000 01E4 DONE equ *
  468    000001E4 4E5E              unlk      a6
  469    000001E6 2E9F              move.l    (sp)+,(sp)
  470    000001E8 4E75              rts
  471
  472    000001EA 003C 0002 range_error ori #2,ccr               force overflow and let system process
  473    000001EE 4E76                    trapv
  474    000001F0 60F2              bra.s done
  475
  476    000001F2 4E75      GLE_ASTEXT_GLE_ASTEXT rts
  477
  478
  479                        END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

BOOTDEFS

Description

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

```
PAGE   1 [3.0] 12/26/84 21:00:47  ASSEMBLY OF BOOTDEFS.TEXT                    *** File name: BOOTDEFS ***

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 [3.0] 12/26/84 21:00:47  ASSEMBLY OF BOOTDEFS.TEXT                    *** File name: BOOTDEFS ***

   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
```

Description

BUB_DVR provides bubble memory card low-level read/write drivers.

```
PASS 1 COMPLETE. ERRORS: 0
    1                       *       BUBBLE MEMORY CARD READ/WRITE DRIVERS
    2                       *       11 JAN 1983
    3                       *
    4                               NOSYMS
    5                       *
    6                               MNAME   BUB_DVR
    7                               DEF     BUB_DVR_BUB_DVR
    8                               DEF     BUB_DVR_BUBGETINFO
    9                               DEF     BUB_DVR_BUBDORESET
   10                               DEF     BUB_DVR_BUBDOREAD
   11                               DEF     BUB_DVR_BUBDOWRITE
   12                               DEF     BUB_DVR_BUBDOISR
   13
```



```
   15                           SRC     MODULE BUB_DVR;
   16                           SRC     EXPORT
   17                           SRC     TYPE
   18                           SRC     BSTATETYPE = (B_IDLE,
   19                           SRC                   B_READING,
   20                           SRC                   B_WRITING);
   21                           SRC
   22                           SRC     BERRORTYPE = (BNOERROR,
   23                           SRC                   BTIMEOUT,
   24                           SRC                   BOPFAILED,
   25                           SRC                   BBADINTERUPT,
   26                           SRC                   BBADSECTOR,
   27                           SRC                   BBADCOUNT,
   28                           SRC                   BNOTBUBBLE,
   29                           SRC                   BBADDATA,
   30                           SRC                   BIOFAIL);
   31                           SRC
   32                           SRC     BBUFP   = ^CHAR;
   33                           SRC     INFOPTR = ^INFOREC;
   34                           SRC     CARDPTR = ^INTEGER;
   35                           SRC
   36                           SRC     BUFREC= RECORD CASE BOOLEAN OF
   37                           SRC               TRUE : (BUFI : INTEGER);
   38                           SRC               FALSE: (BREC : BBUFP);
   39                           SRC             END;
   40                           SRC
   41                           SRC     INFOREC = RECORD
   42                           SRC                 MAXBYTES : INTEGER;
   43                           SRC                 PRIORITY : 0..255;
   44                           SRC                 RUNSTATE : BSTATETYPE;
   45                           SRC                 ERRORCODE: BERRORTYPE;
   46                           SRC
   47                           SRC                 BSTART   : INTEGER;
   48                           SRC                 BBUFFER  : BUFREC;
   49                           SRC                 BCOUNT   : INTEGER;
   50                           SRC                 BRETRY   : INTEGER;
   51                           SRC
   52                           SRC                 BSPAGE   : INTEGER;
   53                           SRC                 BUFSTART : BBUFP;
   54                           SRC                 BUFADDR  : BBUFP;
   55                           SRC                 BUFEND   : BBUFP;
   56                           SRC                 BLOCKSIZE: INTEGER;
   57                           SRC               END;
   58                           SRC
   59                           SRC     PROCEDURE BUBGETINFO(ANYVAR CARD : CARDPTR; ANYVAR INFO : INFOPTR);
   60                           SRC
   61                           SRC     PROCEDURE BUBDORESET(ANYVAR CARD: CARDPTR; ANYVAR INFO : INFOPTR);
   62                           SRC
   63                           SRC     PROCEDURE BUBDOREAD(ANYVAR CARD : CARDPTR; ANYVAR INFO : INFOPTR);
   64                           SRC
   65                           SRC     PROCEDURE BUBDOWRITE(ANYVAR CARD : CARDPTR; ANYVAR INFO : INFOPTR);
   66                           SRC
   67                           SRC     PROCEDURE BUBDOISR(ANYVAR CARD : CARDPTR; ANYVAR INFO : INFOPTR);
   68                           SRC
   69                           SRC END;
   70
```

```
                        *       EQUATES FOR RECORD OFFSETS AND CARD OPERATIONS
                        *
0000 0000 INFOP         EQU     A0
0000 0001 CARDP         EQU     A1
0000 0002 STATUSP       EQU     A2
0000 0002 COMMANDP      EQU     A2
0000 0003 DATAP         EQU     A3
0000 0004 BUFFP         EQU     A4
                        *
                        *       RUNSTATES
0000 0000 B_IDLE        EQU     0
0000 0001 B_READING     EQU     1
0000 0002 B_WRITING     EQU     2
                        *
                        *       ERRORCODES
0000 0000 BNOERROR      EQU     0
0000 0001 BTIMEOUT      EQU     1
0000 0002 BOPFAILED     EQU     2
0000 0003 BBADINTERUPT  EQU     3
0000 0004 BBADSECTOR    EQU     4
0000 0005 BBADCOUNT     EQU     5
0000 0006 BNOTBUBBLE    EQU     6
0000 0007 BBADDATA      EQU     7
0000 0008 BIOFAIL       EQU     8
                        *
                        *       INFO RECORD OFFSETS
0000 0000 MAXBYTES      EQU     0       INTEGER
0000 0004 PRIORITY      EQU     4       WORD
0000 0006 RUNSTATE      EQU     6       WORD
0000 0008 ERRORCODE     EQU     8       WORD
0000 000A BSTART        EQU     10      INTEGER
0000 000E BBUFFER       EQU     14      LONG
0000 0012 BCOUNT        EQU     18      INTEGER
0000 0016 BRETRY        EQU     22      INTEGER
0000 001A BSPAGE        EQU     26      INTEGER
0000 001E BUFSTART      EQU     30      LONG
0000 0022 BUFADDR       EQU     34      LONG
0000 0026 BUFEND        EQU     38      LONG
0000 002A BLOCKSIZE     EQU     42      LONG
                        *
                        *       CARD OFFSETS
0000 0001 CARD_ID       EQU     1
0000 0003 INT_REG       EQU     3
0000 0005 INFO_REG      EQU     5
0000 0009 DATA_REG      EQU     9
0000 000B COMMAND       EQU     11
0000 000B STATUS        EQU     11
                        *
                        *       STATUS BITS
0000 0007 BUSY          EQU     7
0000 0006 OP_DONE       EQU     6
0000 0005 OP_FAIL       EQU     5
0000 0004 TIME_ERR      EQU     4
0000 0003 CORRECTABLE   EQU     3
0000 0002 UNCORRECT     EQU     2
0000 0001 PARITY_ERR    EQU     1
0000 0000 FIFO_AVAIL    EQU     0
```



```
                        *       INTERUPT REG BITS
0000 0007 INT_E         EQU     7       ENABLED
0000 0006 INT_R         EQU     6       REQUESTED
                        *
                        *       BUBBLE COMMAND CODES
0000 0010 WRT_MASKED    EQU     16
0000 0011 INITIALIZE    EQU     17
0000 0012 READ_DATA     EQU     18
0000 0013 WRITE_DATA    EQU     19
0000 0014 READ_SEEK     EQU     20
0000 0015 READ_LOOP_REG EQU     21
0000 0016 WRT_LOOP_REG  EQU     22
0000 0017 WRT_LOOP      EQU     23
0000 0018 READ_FSA      EQU     24
0000 0019 ABORT_CMD     EQU     25
0000 001A WRT_SEEK      EQU     26
0000 001B READ_LOOP     EQU     27
0000 001C READ_CORRECT  EQU     28
0000 001D RESET_FIFO    EQU     29
0000 001E MBM_PURGE     EQU     30
0000 001F SOFT_RESET    EQU     31
0000 0020 CLEAR_INT     EQU     32      CLEAR INTERUPT
                        *
                        *       REG ADDRESS COUNTER VALUES
0000 000A UTILITY       EQU     10      UTILITY REGISTER
0000 000B BLR_LSB       EQU     11      BLOCK LENGTH REG LSB
0000 000C BLR_MSB       EQU     12      BLOCK LENGTH REG MSB
0000 000D ENABLE        EQU     13      ENABLE REGISTER
0000 000E ADDR_LSB      EQU     14      ADDRESS LSB
0000 000F ADDR_MSB      EQU     15      ADDRESS MSB
0000 0000 FIFO          EQU     00      FIFO DATA BUFFER
                        *
                        *       CARD INTERUPT ENABLE/DISABLE
0000 0080 ENABLE_INTS   EQU     128
0000 0000 DISABLE_INTS  EQU     000
                        *
                        *       MISC CONSTANTS
0000 0040 PAGESIZE      EQU     64      CODE EXPLICITY ASSUMES THIS VALUE
0000 0800 MAXPAGES      EQU     2048
```

```
171                         *
172                         *       INTERNAL ROUTINE TO WAIT FOR A COMMAND TO FINISH.
173                         *       HAS A TIME OUT LIMIT
174                         *
175     00000000 0000               DC.W    0
176              0000 0002  COMMAND_DONE    EQU     *
177     00000002 4E56 0000          LINK    A6,#0
178     00000006 226E 0008          MOVEA.L 8(A6),CARDP
179     0000000A 45E9 000B          LEA     STATUS(CARDP),STATUSP
180     0000000E 422E 000C          CLR.B   12(A6)          COMMAND_DONE := FALSE
181     00000012 70FF               MOVEQ   #-1,D0          MAXIMUM COUNT FOR TIMEOUT
182
183     00000014 0829 0006 R76      BTST    #INT_R,INT_REG(CARDP)   WAIT FOR INTERUPT REQUEST
             0003
184     0000001A 56C8 FFF8          DBNE    D0,R76
185     0000001E 6602               BNE.S   R140
186     00000020 6024               BRA.S   CD_EXIT                 NO INTERUPT
187
188     00000022 0812 0007 R140     BTST    #BUSY,(STATUSP)         WAIT FOR NOT BUSY
189     00000026 57C8 FFFA          DBEQ    D0,R140
190     0000002A 6702               BEQ.S   R152
191     0000002C 6018               BRA.S   CD_EXIT                 BUSY DIDN'T GO AWAY
192
193     0000002E 0812 0006 R152     BTST    #OP_DONE,(STATUSP)      CHECK STATUS
194     00000032 6706               BEQ.S   R190                    COMMAND FAILED
195     00000034 1D7C 0001          MOVE.B  #1,12(A6)               COMMAND_DONE := TRUE;
             000C
196
197     0000003A 14BC 0020 R190     MOVE.B  #CLEAR_INT,(COMMANDP)   CLEAR INTERUPT REQUEST
198     0000003E 0829 0006 R196     BTST    #INT_R,INT_REG(CARDP)   WAIT FOR IT TO GO AWAY
             0003
199     00000044 66F8               BNE.S   R196
200
201     00000046 4E5E       CD_EXIT UNLK    A6
202     00000048 205F               MOVEA.L (SP)+,A0
203     0000004A 584F               ADDQ.W  #4,SP
204     0000004C 4ED0               JMP     (A0)
205
```



```
207                         *               EXTERNAL ENTRY POINT TO GET CONFIGURATION INFO
208                         *               ABOUT THE CARD
209                         *               ALSO USED BY BUBDORESET,BUBDOREAD AND BUBDOWRITE
210                         *               TO VALIDATE THE CARD TYPE AND GET MAXBYTES
211                         *               ( THE SIZE OF THE BUBBLE MEMORY UNIT )
212     0000004E 0000               DC.W    0
213              0000 0050  BUB_DVR_BUBGETINFO      EQU     *
214              0000 0050  BUBGETINFO      EQU     *
215     00000050 4E56 0000          LINK    A6,#0
216     00000054 206E 0008          MOVEA.L 8(A6),A0        INFOP
217     00000058 2050               MOVEA.L (A0),INFOP
218     0000005A 226E 000C          MOVEA.L 12(A6),A1       CARDP
219     0000005E 2251               MOVEA.L (A1),CARDP
220
221     00000060 4268 0006          CLR.W   RUNSTATE(INFOP)                 RUNSTATE := B_IDLE
222     00000064 317C 0006          MOVE.W  #BNOTBUBBLE,ERRORCODE(INFOP)    ERRORCODE := BNOTBUBBLE
             0008
223     0000006A 0029 001E          CMPI.B  #30,CARD_ID(CARDP)              CHECK CARD ID
             0001
224     00000070 6648               BNE.S   GI_EXIT
225     00000072 1029 0005          MOVE.B  INFO_REG(CARDP),D0      COPY INFO REG INTO D0
226     00000076 0800 0007          BTST    #7,D0                   CHECK EXTENSION BIT OF INFO_REG
227     0000007A 663E               BNE.S   GI_EXIT
228     0000007C 0800 0004          BTST    #4,D0                   CHECK FSA FIELD OF INFO_REG
229     00000080 6638               BNE.S   GI_EXIT
230     00000082 0800 0003          BTST    #3,D0                   BOTH BITS
231     00000086 6632               BNE.S   GI_EXIT
232                         *                       NO PROBLEMS SO FILL IN MAXBYTES AND PRIORITY
233     00000088 4268 0008          CLR.W   ERRORCODE(INFOP)        ERRORCODE := BNOERROR
234     0000008C 0280 0000          ANDI.L  #7,D0                   GET SIZE FIELD
             0007
235     00000092 5280               ADDQ.L  #1,D0                   ADD 1
236     00000094 720A               MOVEQ   #10,D1                  MULTIPLY BY 1024
237     00000096 E3A0               ASL.L   D1,D0
238     00000098 0829 0005          BTST    #5,INFO_REG(CARDP)      1 MEG OR 4 MEG PARTS
             0005
239     0000009E 6706               BEQ.S   R356
240     000000A0 7209               MOVEQ   #9,D1   4 MEG PARTS
241     000000A2 E3A0               ASL.L   D1,D0   MULTIPLY BY 512
242     000000A4 6002               BRA.S   R366
243     000000A6 EF30       R356    ASL.L   #7,D0   1 MEG PARTS     MULTIPLY BY 128
244     000000A8 2080       R366    MOVE.L  D0,(INFOP)      SET MAXBYTES
245     000000AA 1029 0003          MOVE.B  INT_REG(CARDP),D0       GET PRIORITY
246     000000AE E848               LSR.W   #4,D0   FROM FIELD OF INTERUPT REG ON CARD
247     000000B0 0240 0003          ANDI.W  #3,D0
248     000000B4 5640               ADDQ.W  #3,D0
249     000000B6 3140 0004          MOVE.W  D0,PRIORITY(INFOP)
250     000000BA 4E5E       GI_EXIT UNLK    A6
251     000000BC 205F               MOVEA.L (SP)+,A0
252     000000BE 504F               ADDQ.W  #8,SP
253     000000C0 4ED0               JMP     (A0)
254
```

```
                        *
                        *               INTERNAL ROUTINE TO LOAD THE BUBBLE CONTROLER
                        *               PARAMETER REGISTERS PRIOR TO MAJOR OPERATIONS
                        *
000000C2 0000           DC.W    0
         0000 00C4 INITIALREGS  EQU     *
000000C4 4E56 0000      LINK    A6,#0
000000C8 226E 0010      MOVEA.L 16(A6),CARDP
000000CC 206E 000C      MOVEA.L 12(A6),INFOP
000000D0 47E9 0009      LEA     DATA_REG(CARDP),DATAP
000000D4 137C 000B      MOVE.B  #BLR_LSB,COMMAND(CARDP) point RAC at BLR_LSB
         0008
                        *
000000D8 16A8 002D      MOVE.B  BLOCKSIZE+3(INFOP),(DATAP)      set LSB of BLOCK LENGTH REG.
000000DC 1028 002C      MOVE.B  BLOCKSIZE+2(INFOP),D0   get msb
000000E0 D07C 0010      ADD.W   #16,D0                  set bit to use 2 FSA channels
000000E4 1680           MOVE.B  D0,(DATAP)              set MSB of BLOCK LENGTH REG.
                        *
000000E6 7200           MOVEQ   #0,D1           clear MFBTR bit enable burst on last page
000000E8 D2AE 0008      ADD.L   8(A6),D1 add in operation peculiar bits from parameter list
000000EC 1681           MOVE.B  D1,(DATAP)              set ENABLE REG.
                        *
000000EE 16A8 001D      MOVE.B  BSPAGE+3(INFOP),(DATAP)         set LSB of ADDRESS REG.
000000F2 16A8 001C      MOVE.B  BSPAGE+2(INFOP),(DATAP)         set MSB of ADDRESS REG.
                        *
000000F6 4E5E           UNLK    A6
000000F8 205F           MOVEA.L (SP)+,A0
000000FA DEFC 000C      ADDA.W  #12,SP
000000FE 4ED0           JMP     (A0)
                        *
```



```
                        *
                        *               EXTERNAL ENTRY POINT TO "INITIALIZE" THE
                        *               BUBBLE MEMORY CONTROLER FOR READ/WRITE OPERATIONS
                        *               ALSO USED TO ABORT ANY CURRENT READ/WRITE OPERATION
                        *               (A6)
                        *               -4      BITCOUNT
                        *               -8      BYTECOUNT
                        *              -12      ABORT RETRY COUNTER
                        *              -16      SAVED CARD ADDRESS
                        *              -20      SAVED INFO RECORD ADDRESS
                        *
00000102 0000           DC.W    0
         0000 0104 BUB_DVR_BUBDORESET   EQU     *
00000104 4E56 FFEC      LINK    A6,#-20
00000108 2F2E 000C      MOVE.L  12(A6),-(SP)    CARD
0000010C 2F2E 0008      MOVE.L  8(A6),-(SP)     INFO
00000110 4EBA FF3E      JSR     BUBGETINFO      FILL IN INFO FIELDS AND CHECK CARD TYPE
00000114 206E 0008      MOVEA.L 8(A6),A0        GET INFOREC POINTER
00000118 2050           MOVEA.L (A0),INFOP
0000011A 4A68 0008      TST.W   ERRORCODE(INFOP)
0000011E 6600 012C      BNE     RS_EXIT
00000122 2D48 FFEC      MOVE.L  INFOP,-20(A6)   SAVE INFOP
00000126 226E 000C      MOVEA.L 12(A6),A1
0000012A 2251           MOVEA.L (A1),CARDP
0000012C 2D49 FFF0      MOVE.L  CARDP,-16(A6)   SAVE CARDP
                        *
00000130 137C 0000      MOVE.B  #0,CARD_ID(CARDP)       RESET THE CARD
         0001
00000136 203C 0000      MOVE.L   #300,D0         KILL AT LEAST 200 MICRO SECONDS         (rdq)
         012C
0000013C 51C8 FFFE WAIT200 DBF  D0,WAIT200      *** TIMEING LOOP ***
                        *
00000140 1D7C 0003      MOVE.B  #3,-12(A6)      SET RETRY COUNT FOR ABORT COMMAND
         FFF4
00000146 45E9 000B R700 LEA     COMMAND(CARDP),COMMANDP
                        *                               ENABLE INTERUPTS ON OPDONE
0000014A 14BC 000D      MOVE.B  #ENABLE,(COMMANDP)      POINT RAC AT ENABLE REG
0000014E 137C 0001      MOVE.B  #1,DATA_REG(CARDP)      SET ENABLE REG
         0009
                        *
00000154 14BC 0019      MOVE.B  #ABORT_CMD,(COMMANDP)   ABORT ANY CURRENT OPERATION
00000158 558F           SUBQ.L  #2,SP           SPACE FOR FUNCTION VALUE
0000015A 2F09           MOVE.L  CARDP,-(SP)     CARD^
0000015C 4EBA FEA4      JSR     COMMAND_DONE    WAIT FOR IT TO FINISH
00000160 4A1F           TST.B   (SP)+   DID IT FINISH OK ?
00000162 660E           BNE.S   R800            IF FINISHED OK THEN PROCEEDE
00000164 532E FFF4      SUBQ.B  #1,-12(A6)      DECRIMENT RETRY COUNTER
00000168 6700 00D8      BEQ     R1176           IF COUNT DONE THEN GIVE UP
0000016C 226E FFF0      MOVEA.L -16(A6),CARDP   RELOAD CARDP
00000170 60D4           BRA     R700            TRY AGAIN
                        *
00000172 206E FFEC R800 MOVEA.L -20(A6),INFOP           RETRIEVE INFOP
00000176 2E28 0000      MOVE.L  MAXBYTES(INFOP),D7      BSPAGE := MAXBYTES DIV 64 - 1;
0000017A EC87           ASR.L   #6,D7   DIV 64
0000017C 5387           SUBQ.L  #1,D7   - 1
0000017E 2147 001A      MOVE.L  D7,BSPAGE(INFOP)
                        *
```

```
 339  00000182 217C 0000          MOVE.L  #1,BLOCKSIZE(INFOP)     BLOCKSIZE := 1 PAGE
              0001 00EA
 340  0000018A 2F2E FFF0          MOVE.L  -16(A6),-(SP)   CARD^
 341  0000018E 2F08               MOVE.L  INFOP,-(SP)     INFO^
 342  00000190 2F3C 0000          MOVE.L  #$21,-(SP)      ENABLE RCD AND OPDONE FUNCTIONS
              0021
 343  00000196 4EBA FF2C          JSR     INITIALREGS     FIXUP THE PARAMETER REGS
 344
 345  0000019A 226E FFF0          MOVEA.L -16(A6),CARDP   RETRIEVE CARDP
 346  0000019E 45E9 000B          LEA     STATUS(CARDP),STATUSP   { STATUSP is same as COMMANDP }
 347  000001A2 137C 0011          MOVE.B  #INITIALIZE,COMMAND(CARDP)
              000B
 348  000001A8 0829 0006 R922     BTST    #INT_R,INT_REG(CARDP)   WAIT FOR INTERUPT
              0003
 349  000001AE 67F8               BEQ     R922                    (ON FIFO HALF FULL)
 350  000001B0 0812 0007 R944     BTST    #BUSY,(STATUSP)         WAIT FOR NOT BUSY
 351  000001B4 66FA               BNE     R944
 352  000001B6 0812 0006          BTST    #OP_DONE,(COMMANDP)     DID IT WORK ?
 353  000001BA 6700 007A          BEQ     R1162
 354  000001BE 2F09               MOVE.L  CARDP,-(SP)     CARD^
 355  000001C0 226E 0008          MOVEA.L 8(A6),A1
 356  000001C4 2F11               MOVE.L  (A1),-(SP)      INFO^
 357  000001C6 2F3C 0000          MOVE.L  #$21,-(SP)      ENABLE RCD AND OPDONE FUNCTIONS
              0021
 358  000001CC 4EBA FEF6          JSR     INITIALREGS     FIX PARAMETER REGS
 359                      *                               { THIS ALSO STOPS CURRENT INTERUPT }
 360  000001D0 226E FFF0          MOVEA.L -16(A6),CARDP   RETRIEVE CARDP
 361  000001D4 45E9 000B          LEA     STATUS(CARDP),STATUSP   { STATUSP is same as COMMANDP }
 362  000001D8 14BC 0015          MOVE.B  #READ_LOOP_REG,(COMMANDP)       READ BOOT LOOP REGISTER
 363  000001DC 0829 0006 R1018    BTST    #INT_R,INT_REG(CARDP)           WAIT FOR INTERUPT
              0003
 364  000001E2 67F8               BEQ     R1018
 365  000001E4 42AE FFFC R1040    CLR.L   -4(A6)          BITCOUNT := 0
 366  000001E8 42AE FFF8          CLR.L   -8(A6)          BYTECOUNT := 0
 367  000001EC 0829 0007 R1048    BTST    #BUSY,STATUS(CARDP)     WHILE BUSY DO
              000B
 368  000001F2 6724               BEQ.S   R1124
 369  000001F4 0829 0000 R1068    BTST    #FIFO_AVAIL,STATUS(CARDP)       WHILE FIFO_AVAIL DO
              000B
 370  000001FA 67F0               BEQ     R1048           IF FIFO EMPTY THEN CHECK BUSY
 371  000001FC 52AE FFF8          ADDQ.L  #1,-8(A6)       INCREMENT BYTECOUNT
 372  00000200 1029 0009          MOVE.B  DATA_REG(CARDP),D0      READ THE DATA BYTE
 373                      *                               COUNT THE 1 BITS IN LOW BYTE OF D0
 374  00000204 7200               MOVEQ   #0,D1   COUNT := 0
 375  00000206 7407               MOVEQ   #7,D2   LOOP COUNTER
 376  00000208 7600               MOVEQ   #0,D3   ZERO CONSTANT
 377  0000020A E308       COUNTL  LSL.B   #1,D0   SHIFT HI BIT TO X
 378  0000020C D303               ADDX.B  D3,D1   ADD 0 + X + COUNT { WILL NEVER BE MORE THAN 8 }
 379  0000020E 51CA FFFA          DBRA    D2,COUNTL
 380
 381  00000212 D3AE FFFC          ADD.L   D1,-4(A6)       ADD COUNT TO BIT COUNT
 382  00000216 60DC               BRA     R1068   END WHILE FIFO_AVAIL
 383
 384  00000218 7228       R1124   MOVEQ   #40,D1          MUST HAVE EXACTLY 40 BYTES
 385  0000021A B2AE FFF8          CMP.L   -8(A6),D1
 386  0000021E 660A               BNE.S   R1150
```



```
 387  00000220 0CAE 0000          CMPI.L  #270,-4(A6)      MUST HAVE EXACTLY 270 BITS
              010E FFFC
 388  00000228 6722               BEQ.S   RS_EXIT
 389  0000022A 206E FFEC R1150    MOVEA.L -20(A6),INFOP   WRONG BYTE / BIT COUNT
 390  0000022E 317C 0002          MOVE.W  #BOPFAILED,ERRORCODE(INFOP)
              0008
 391  00000234 6016               BRA.S   RS_EXIT
 392  00000236 206E FFEC R1162    MOVEA.L -20(A6),INFOP   INITIALIZE OPERATION FAILED
 393  0000023A 317C 0002          MOVE.W  #BOPFAILED,ERRORCODE(INFOP)
              0008
 394  00000240 600A               BRA.S   RS_EXIT
 395  00000242 206E FFEC R1176    MOVEA.L -20(A6),INFOP   FAILED TO RESPOND TO ABORT COMMAND
 396  00000246 317C 0001          MOVE.W  #BTIMEOUT,ERRORCODE(INFOP)
              0008
 397  0000024C 4E5E       RS_EXIT UNLK A6
 398  0000024E 205F               MOVEA.L (SP)+,A0
 399  00000250 504F               ADDQ.W  #8,SP
 400  00000252 4ED0               JMP (A0)
 401
```

```
                        *
                        *               FINAL STAGE OF BUBDOREAD
                        *               ALSO USED BY BUBDOISR TO RESTART A FAILED READ OPERATION
                        *               (A6)
                        *               -4      SAVED CARD ADDRESS
                        *               -8      SAVED INFO RECORD ADDRESS
                        *
00000254 0000                   DC.W    0
         0000 0256 STARTREAD    EQU     *
00000256 4E56 FFF8              LINK    A6,#-8
0000025A 226E 000C              MOVEA.L 12(A6),A1       CARD
0000025E 2D51 FFFC              MOVE.L  (A1),-4(A6)
00000262 206E 0008              MOVEA.L 8(A6),A0        INFO
00000266 2D50 FFF8              MOVE.L  (A0),-8(A6)
0000026A 2050                   MOVEA.L (A0),INFOP      INFO^
0000026C 4268 0006              CLR.W   RUNSTATE(INFOP)         RUNSTATE := B_IDLE
00000270 4268 0008              CLR.W   ERRORCODE(INFOP)        ERRORCODE := BNOERROR
00000274 226E FFFC              MOVEA.L -4(A6),CARDP    CARD^
00000278 137C 000D              MOVE.B  #ENABLE,COMMAND(CARDP)  SET RAC TO ENABLE REG.
         000B
0000027E 137C 0001              MOVE.B  #1,DATA_REG(CARDP)      ENABLE INTERUPT ON OPDONE
         0009
00000284 137C 001D              MOVE.B  #RESET_FIFO,COMMAND(CARDP)      CLEAR THE FIFO
         000B
0000028A 558F                   SUBQ.L  #2,SP           SPACE FOR FUNCTION VALUE
0000028C 2F09                   MOVE.L  CARDP,-(SP)     CARD^
0000028E 4EBA FD72              JSR     COMMAND_DONE                    WAIT FOR IT
00000292 4A1F                   TST.B   (SP)+           DID IT WORK ?
00000294 6734                   BEQ.S   R1332
00000296 2F2E FFFC              MOVE.L  -4(A6),-(SP)    CARD^
0000029A 2F2E FFF8              MOVE.L  -8(A6),-(SP)    INFO^
0000029E 2F3C 0000              MOVE.L  #$21,-(SP)      ENABLE RCD AND OPDONE FUNCTIONS
         0021
000002A4 4EBA FE1E              JSR     INITIALREGS     FIX CONTROL REGS.
000002A8 206E FFF8              MOVEA.L -8(A6),INFOP
000002AC 2168 001E              MOVE.L  BUFSTART(INFOP),BUFADDR(INFOP)  SET BUFFER ADDRESS
         0022
000002B2 317C 0001              MOVE.W  #B_READING,RUNSTATE(INFOP)      SHOW NOW READING
         0006
000002B8 226E FFFC              MOVEA.L -4(A6),CARDP
000002BC 137C 0080              MOVE.B  #ENABLE_INTS,INT_REG(CARDP)     ENABLE CARD INTERUPTS
         0003
000002C2 137C 0012              MOVE.B  #READ_DATA,COMMAND(CARDP)       START READ OPERTATION
         000B
000002C8 600A                   BRA.S   SR_EXIT
000002CA 206E FFF8 R1332        MOVEA.L -8(A6),INFOP            RESET FIFO FAILED
000002CE 317C 0001              MOVE.W  #BTIMEOUT,ERRORCODE(INFOP)
         0008
000002D4 4E5E          SR_EXIT  UNLK    A6
000002D6 205F                   MOVEA.L (SP)+,A0
000002D8 504F                   ADDQ.W  #8,SP
000002DA 4ED0                   JMP     (A0)
```



```
                        *
                        *               EXTERNAL ENTRY POINT TO START DATA READ OPERATIONS
                        *
000002DC 0000                   DC.W    0
         0000 02DE BUB_DVR_BUBDOREAD    EQU     *
000002DE 4E56 0000              LINK    A6,#0
000002E2 2F2E 000C              MOVE.L  12(A6),-(SP)    CARD
000002E6 2F2E 0008              MOVE.L  8(A6),-(SP)     INFO
000002EA 4EBA FD64              JSR     BUBGETINFO      FILL INFO FIELDS AND CHECK THE CARD TYPE
000002EE 206E 0008              MOVEA.L 8(A6),A0
000002F2 2050                   MOVEA.L (A0),INFOP      INFO^
000002F4 4A68 0008              TST.W   ERRORCODE(INFOP)        IF ERRORCODE<>BNOERROR
000002F8 6600 0076              BNE     RD_EXIT                 THEN QUIT
000002FC 226E 000C              MOVEA.L 12(A6),A1
00000300 2251                   MOVEA.L (A1),CARDP      CARD^

00000302 2028 000A              MOVE.L  BSTART(INFOP),D0        CHECK STARTING POSITION (BYTE)
00000306 6B04                   BMI.S   R1420           CAN'T BE NEGATIVE
00000308 4A00                   TST.B   D0              MUST BE MULTIPLE OF 256
0000030A 670A                   BEQ.S   R1428
0000030C 317C 0004 R1420        MOVE.W  #BBADSECTOR,ERRORCODE(INFOP)
         0008
00000312 6000 005C              BRA     RD_EXIT

00000316 2228 0012 R1428        MOVE.L  BCOUNT(INFOP),D1        CHECK BYTE COUNT
0000031A 6F06                   BLE.S   R1450           MUST BE GREATER THAN ZERO
0000031C D081                   ADD.L   D1,D0           BCOUNT + BSTART
0000031E B090                   CMP.L   (INFOP),D0      CAN'T BE GREATER THAN MAXBYTES
00000320 6F08                   BLE.S   R1456
00000322 317C 0005 R1450        MOVE.W  #BBADCOUNT,ERRORCODE(INFOP)
         0008
00000328 6046                   BRA.S   RD_EXIT
0000032A 137C 0000 R1456        MOVE.B  #DISABLE_INTS,INT_REG(CARDP)    DISABLE CARD INTERUPTS
         0003
00000330 2141 0026              MOVE.L  D1,BUFEND(INFOP)        SET COUNT IN BUFEND
                        *               CALCULATE NUMBER OF PAGES FOR THIS OPERATION
                        *                               PAGE SIZE IS ASSUMED TO BE 64 BYTES
00000334 D2BC 0000              ADD.L   #63,D1  ROUND UP BYTECOUNT
         003F
0000033A EC81                   ASR.L   #6,D1   DIVIDE BY 64    D1 IS NOW NUMBER OF PAGES
0000033C 0C81 0000              CMPI.L  #MAXPAGES,D1    CLIP IT AT MAXPAGES
         0800
00000342 6D06                   BLT.S   R1480
00000344 223C 0000              MOVE.L  #MAXPAGES,D1
         0800
0000034A 2141 002A R1480        MOVE.L  D1,BLOCKSIZE(INFOP)
                        *               CALCULATE START PAGE FOR THIS OPERATION
0000034E 2028 000A              MOVE.L  BSTART(INFOP),D0
00000352 EC80                   ASR.L   #6,D0   DIVIDE BY 64
00000354 2140 001A              MOVE.L  D0,BSPAGE(INFOP)
00000358 2028 000E              MOVE.L  BBUFFER(INFOP),D0   GET CURRENT START OF BUFFER
0000035C 2140 001E              MOVE.L  D0,BUFSTART(INFOP)  MARK START OF BUFFER
00000360 D1A8 0026              ADD.L   D0,BUFEND(INFOP)    MARK END OF BUFFER (ADD ADDRESS TO COUNT)
00000364 2F2E 000C              MOVE.L  12(A6),-(SP)    CARD
00000368 2F2E 0008              MOVE.L  8(A6),-(SP)     INFO
0000036C 4EBA FEE8              JSR     STARTREAD       FINISH UP AND ISSUE THE READ COMMAND
00000370 4E5E          RD_EXIT  UNLK    A6
```

```
  499    00000372 205F              MOVEA.L (SP)+,A0
  500    00000374 504F              ADDQ.W  #8,SP
  501    00000376 4ED0              JMP     (A0)
  502
```



```
  504                       *
  505                       *               FINAL STAGE OF BUBDOWRITE
  506                       *               ALSO USED BY BUBDOISR TO RESTART A FAILED WRITE OPERATION
  507                       *               (A6)
  508                       *               -4      SAVED CARD ADDRESS
  509                       *               -8      SAVED INFO RECORD ADDRESS
  510                       *
  511    00000378 0000              DC.W    0
  512              0000 037A STARTWRITE      EQU     *
  513    0000037A 4E56 FFF8         LINK    A6,#-8
  514    0000037E 206E 000C         MOVEA.L 12(A6),A0       CARD
  515    00000382 2D50 FFFC         MOVE.L  (A0),-4(A6)     SAVE CARD^
  516    00000386 206E 0008         MOVEA.L 8(A6),A0        INFO
  517    0000038A 2050              MOVEA.L (A0),INFOP      INFO^
  518    0000038C 2D48 FFF8         MOVE.L  INFOP,-8(A6)    SAVE INFO^
  519    00000390 4268 0006         CLR.W   RUNSTATE(INFOP)         RUNSTATE := B_IDLE
  520    00000394 4268 0008         CLR.W   ERRORCODE(INFOP)        ERRORCODE:= BNOERROR
  521    00000398 2F2E FFFC         MOVE.L  -4(A6),-(SP)    CARD^
  522    0000039C 2F08              MOVE.L  INFOP,-(SP)     INFO^
  523    0000039E 2F3C 0000         MOVE.L  #$21,-(SP)      ENABLE RCD AND OPDONE FUNCTIONS
                  0021
  524    000003A4 4EBA FD1E         JSR     INITIALREGS     FIX CONTROL REGS.
  525    000003A8 226E FFFC         MOVEA.L -4(A6),CARDP
  526    000003AC 137C 001D         MOVE.B  #RESET_FIFO,COMMAND(CARDP)      CLEAR/RESET THE FIFO
                  000B
  527    000003B2 598F              SUBQ.L  #2,SP           SPACE FOR FUNCTION VALUE
  528    000003B4 2F09              MOVE.L  CARDP,-(SP)     CARD^
  529    000003B6 4EBA FC4A         JSR     COMMAND_DONE    WAIT FOR FIFO TO RESET
  530    000003BA 4A1F              TST.B   (SP)+           DID IT WORK ?
  531    000003BC 6702              BEQ.S   R1734
  532    000003BE 206E FFF8         MOVEA.L -8(A6),INFOP    OK SO CONTINUE
  533    000003C2 2168 001E         MOVE.L  BUFSTART(INFOP),BUFADDR(INFOP)  SET BUFFER ADDRESS
                  0002
  534    000003C8 317C 0002         MOVE.W  #B_WRITING,RUNSTATE(INFOP)      SHOW NOW WRITING
                  0006
  535    000003CE 226E FFFC         MOVEA.L -4(A6),CARDP
  536    000003D2 137C 0080         MOVE.B  #ENABLE_INTS,INT_REG(CARDP)     ENABLE CARD INTERUPTS
                  0003
  537    000003D8 137C 0013         MOVE.B  #WRITE_DATA,COMMAND(CARDP)      START THE OPERATION
                  000B
  538    000003DE 600A              BRA.S   SW_EXIT
  539
  540    000003E0 206E FFF8 R1734   MOVEA.L -8(A6),INFOP            FIFO RESET FAILED
  541    000003E4 317C 0001         MOVE.W  #BTIMEOUT,ERRORCODE(INFOP)
                  0008
  542
  543    000003EA 4E5E      SW_EXIT UNLK    A6
  544    000003EC 205F              MOVEA.L (SP)+,A0
  545    000003EE 504F              ADDQ.W  #8,SP
  546    000003F0 4ED0              JMP     (A0)
  547
```

```
PAGE  15 [3.0] 12/26/84 21:43:09  ASSEMBLY OF BUB_DVR.TEXT                                  *** File name: BUB_DVR ***

549                             *
550                             *               EXTERNAL ENTRY POINT TO START DATA WRITE OPERATIONS
551                             *               (A6)
552                             *               -4      UNUSED          (COMPILER TEMP OPTIMIZED OUT)
553                             *               -8      SAVED CARD ADDRESS
554                             *              -12      SAVED INFO RECORD ADDRESS
555                             *
556     000003F2 0000                   DC.W    0
557              0000 03F4 BUB_DVR_BUBDOWRITE   EQU     *
558     000003F4 4E56 FFF4              LINK    A6,#-12
559     000003F8 2F2E 000C              MOVE.L  12(A6),-(SP)    CARD
560     000003FC 2F2E 0008              MOVE.L  8(A6),-(SP)     INFO
561     00000400 4EBA FC4E              JSR     BUBGETINFO      FILL IN INFO FIELDS AND CHECK THE CARD TYPE
562     00000404 206E 0008              MOVEA.L 8(A6),A0
563     00000408 2050                   MOVEA.L (A0),INFOP
564     0000040A 4A68 0008              TST.W   ERRORCODE(INFOP)        IF ERRORCODE <> BNOERROR
565     0000040E 6600 0088              BNE     WT_EXIT                 THEN EXIT
566     00000412 226E 000C              MOVEA.L 12(A6),A1
567     00000416 2D51 FFF8              MOVE.L  (A1),-8(A6)     CARD^
568     0000041A 206E 0008              MOVEA.L 8(A6),A0
569     0000041E 2050                   MOVEA.L (A0),INFOP      INFO^
570     00000420 2D48 FFF4              MOVE.L  INFOP,-12(A6)
571     00000424 2028 000A              MOVE.L  BSTART(INFOP),D0        CHECK STARTING POSITION (BYTE)
572     00000428 6B04                   BMI.S   R1820                   CAN'T BE NEGATIVE
573     0000042A 4A00                   TST.B   D0                      MUST BE MULTIPLE OF 256
574     0000042C 670A                   BEQ.S   R1830
575     0000042E 317C 0004 R1820        MOVE.W  #BBADSECTOR,ERRORCODE(INFOP)
                 0008
576     00000434 6000 0062              BRA     WT_EXIT
577
578     00000438 2228 0012 R1830        MOVE.L  BCOUNT(INFOP),D1        CHECK BYTECOUNT
579     0000043C 6F0A                   BLE.S   R1860                   MUST BE GREATER THAN ZERO
580     0000043E 4A01                   TST.B   D1
581     00000440 6606                   BNE.S   R1860                   MUST BE MULTIPLE OF 256
582     00000442 D081                   ADD.L   D1,D0
583     00000444 B090                   CMP.L   (A0),D0         IF (BCOUNT+BSTART)<=MAXBYTES
584     00000446 6F0A                   BLE.S   R1876           THEN CONTINUE
585     00000448 317C 0005 R1860        MOVE.W  #BBADCOUNT,ERRORCODE(INFOP)
                 0008
586     0000044E 6000 0048              BRA     WT_EXIT
587
588     00000452 226E FFF8 R1876        MOVEA.L -8(A6),CARDP    CARD^
589     00000456 137C 0000              MOVE.B  #DISABLE_INTS,INT_REG(CARDP)    DISABLE CARD INTERUPTS
                 0003
590                             *               CALCULATE NUMBER OF PAGES FOR THIS OPERATION
591                             *                               PAGE SIZE IS ASSUMED TO BE 64 BYTES
592     0000045C EC81                   ASR.L   #6,D1   DIVIDE BY 64    D1 IS NOW NUMBER OF PAGES
593     0000045E 0C81 0000              CMPI.L  #MAXPAGES,D1    CLIP IT AT MAXPAGES
                 0800
594     00000464 6D06                   BLT.S   R1880
595     00000466 223C 0000              MOVE.L  #MAXPAGES,D1
                 0800
596     0000046C 2141 002A R1880        MOVE.L  D1,BLOCKSIZE(INFOP)
597                             *                               FOR WRITE OPS, BUFEND IS END OF SEGMENT
598     00000470 ED81                   ASL.L   #6,D1           CONVERT BLOCKSIZE BACK TO BYTES
599     00000472 2141 0026              MOVE.L  D1,BUFEND(INFOP)        SAVE IT IN BUFEND
600
```

```
PAGE  16 [3.0] 12/26/84 21:43:09  ASSEMBLY OF BUB_DVR.TEXT                                  *** File name: BUB_DVR ***

601                             *               CALCULATE START PAGE FOR THIS OPERATION
602     00000476 2028 000A              MOVE.L  BSTART(INFOP),D0
603     0000047A EC80                   ASR.L   #6,D0   DIVIDE BY 64
604     0000047C 2140 001A              MOVE.L  D0,BSPAGE(INFOP)
605
606     00000480 2028 000E              MOVE.L  BBUFFER(INFOP),D0   GET CURRENT START OF BUFFER
607     00000484 2140 001E              MOVE.L  D0,BUFSTART(INFOP)  MARK START OF BUFFER
608     00000488 D1A8 0026              ADD.L   D0,BUFEND(INFOP)    MARK END OF BUFFER (ADD ADDRESS TO SIZE)
609     0000048C 2F2E 000C              MOVE.L  12(A6),-(SP)    CARD
610     00000490 2F2E 0008              MOVE.L  8(A6),-(SP)     INFO
611     00000494 4EBA FEE4              JSR     STARTWRITE      FINISH UP AND ISSUE THE WRITE COMMAND
612     00000498 4E5E           WT_EXIT UNLK    A6
613     0000049A 205F                   MOVEA.L (SP)+,A0
614     0000049C 504F                   ADDQ.W  #8,SP
615     0000049E 4ED0                   JMP     (A0)
616
```

```
                        *
                        *                       INTERUPT SERVICE ROUTINE
                        *
000004A0 0000           DC.W    0
         0000 04A2 BUB_DVR_BUBDOISR      EQU     *
000004A2 4E56 0000      LINK    A6,#0
000004A6 226E 000C      MOVEA.L 12(A6),A1       GET CARD ADDRESS
000004AA 2251           MOVEA.L (A1),CARDP              CARD BASE ADDRESS
000004AC 45E9 000B      LEA     STATUS(CARDP),STATUSP   STATUS REG ADDRESS
000004B0 47E9 0009      LEA     DATA_REG(CARDP),DATAP   DATA REG ADDRESS

000004B4 206E 0008      MOVEA.L 8(A6),A0        GET INFOREC ADDRESS
000004B8 2050           MOVEA.L (A0),INFOP              INFOREC BASE ADDRESS
000004BA 2968 0022      MOVEA.L BUFADDR(INFOP),BUFFP    BUFFER ADDRESS

000004BE 0C68 0001      CMPI.W  #B_READING,RUNSTATE(A0) JUMP ACCORDING TO RUNSTATE
         0006
000004C4 671C           BEQ.S   READING
000004C6 6E00 011E      BGT     WRITING

                        *       IN B_IDLE STATE ... HAVE UNEXPECTED INTERUPT
000004CA 317C 0003 IDLE MOVE.W  #BBADINTERUPT,ERRORCODE(INFOP)  set ERRORCODE
         0008
000004D0 137C 0000      MOVE.B  #DISABLE_INTS,INT_REG(CARDP)    disable interupt enable
         0003
000004D6 0829 0007 IDLE1 BTST   #INT_E,INT_REG(CARDP)   wait for enable bit to clear
         0003
000004DC 66F8           BNE.S   IDLE1
000004DE 6000 01CE      BRA     ISRDONE                 all done

                        *       IN B_READING    ... HAVE DATA OR END OF OPERATION
000004E2 0812 0007 READING BTST  #BUSY,(STATUSP)
000004E6 672A           BEQ.S   R2232
                        *                               BUSY SO HAVE DATA TO READ
000004E8 2028 0026      MOVE.L  BUFEND(INFOP),D0        GET END OF BUFFER ADDRESS
000004EC B08C      R2132 CMP.L  BUFFP,D0          CHECK BUFFER AGAINST END OF BUFFER
000004EE 6712           BEQ.S   R2143
000004F0 0812 0000      BTST    #FIFO_AVAIL,(STATUSP)   CHECK FIFO
000004F4 6704           BEQ.S   R2135
000004F6 18D3      R2133 MOVE.B  (DATAP),(BUFFP)+        READ THE DATA
000004F8 60F2           BRA.S   R2132

000004FA 214C 0022 R2135 MOVE.L  BUFFP,BUFADDR(INFOP)   PUT BACK BUFFER ADDRESS
000004FE 6000 01AE      BRA     ISRDONE                 THEN EXIT

00000502 214C 0022 R2143 MOVE.L  BUFFP,BUFADDR(INFOP)   PUT BACK BUFFER ADDRESS
                        *                               THEN DISCARD UNWANTED DATA
00000506 0812 0000 R2196 BTST    #FIFO_AVAIL,(STATUSP)  CHECK FIFO
0000050A 6700 01A2      BEQ     ISRDONE                 EXIT IF FIFO IS EMPTY
0000050E 1013           MOVE.B  (DATAP),D0              READ & DISCARD
00000510 60F4           BRA.S   R2196
                        *                               NOT BUSY SO END OF OPERATION
                        *       DID ANYTHING GO WRONG ?
00000512 0812 0004 R2232 BTST    #TIME_ERR,(STATUSP)
00000516 6612           BNE.S   R2240
00000518 0812 0005      BTST    #OP_FAIL,(STATUSP)
0000051C 660C           BNE.S   R2240
```



```
0000051E 0812 0002      BTST    #UNCORRECT,(STATUSP)
00000522 673E           BEQ.S   R2348
00000524 317C 0007      MOVE.W  #BBADDATA,ERRORCODE(INFOP)      SET BADDATA ERROR
         0008
                        *       SOMETHING WENT WRONG SO TRY TO RESTART THE CURRENT REQUEST
0000052A 14BC 0020 R2240 MOVE.B  #CLEAR_INT,(COMMANDP)  CLEAR INTERUPT REQUEST
0000052E 0829 0006 R2280 BTST    #INT_R,INT_REG(CARDP)  WAIT FOR IT TO GO AWAY
         0003
00000534 66F8           BNE.S   R2280

00000536 53A8 0016      SUBQ.L  #1,BRETRY(INFOP)        DECRIMENT RETRY COUNTER
0000053A 6D10           BLT.S   R2334
0000053C 2F2E 000C      MOVE.L  12(A6),-(SP)    CARD
00000540 2F2E 0008      MOVE.L  8(A6),-(SP)     INFO
00000544 4EBA FD10      JSR     STARTREAD       RESTART THE CURRENT REQUEST
00000548 6000 0164      BRA     ISRDONE

                        *       TOO MANY RETRIES
0000054C 4268 0006 R2334 CLR.W   RUNSTATE(INFOP)  SET RUNSTATE TO B_IDLE
00000550 4A68 0008      TST.W   ERRORCODE(INFOP) IS ERRORCODE ALREADY SET ?
00000554 6600 0158      BNE     ISRDONE
00000558 317C 0008      MOVE.W  #BIOFAIL,ERRORCODE(INFOP)
         0008
0000055E 6000 014E      BRA     ISRDONE

                        *       OPERATION COMPLETED OK
                        *               READ ANY DATA LEFT IN THE FIFO
00000562 2028 0026 R2348 MOVE.L  BUFEND(INFOP),D0       GET END OF BUFFER ADDRESS
00000566 B08C      R2350 CMP.L  BUFFP,D0          CHECK BUFFER ADDRESS AGAINST END OF BUFFER
00000568 670A           BEQ.S   R2412
0000056A 0812 0000      BTST    #FIFO_AVAIL,(STATUSP)   CHECK THE FIFO
0000056E 670E           BEQ.S   R2440
00000570 18D3           MOVE.B  (DATAP),(BUFFP)+
00000572 60F2           BRA.S   R2350
                        *               DISCARD THE REST OF THE DATA
00000574 0812 0000 R2412 BTST    #FIFO_AVAIL,(STATUSP)   CHECK THE FIFO
00000578 6704           BEQ.S   R2440
0000057A 1213           MOVE.B  (DATAP),D1      DISCARD THE DATA
0000057C 60F6           BRA.S   R2412

0000057E 214C 0022 R2440 MOVE.L  BUFFP,BUFADDR(INFOP)   PUT BACK THE BUFFER ADDRESS
00000582 14BC 0020      MOVE.B  #CLEAR_INT,(COMMANDP)   CLEAR THE INTERUPT REQUEST
00000586 0829 0006 R2454 BTST    #INT_R,INT_REG(CARDP)  WAIT FOR IT TO GO AWAY
         0003
0000058C 66F8           BNE.S   R2454

0000058E B08C           CMP.L   BUFFP,D0        IF ALL DATA READ
00000590 6700 0118      BEQ     R3008           THEN OPERATION COMPLETED
00000594 2028 002A      MOVE.L  BLOCKSIZE(INFOP),D0             INCREMENT START PAGE
00000598 D1A8 001A      ADD.L   D0,BSPAGE(INFOP)

0000059C 2028 0022      MOVE.L  BUFADDR(INFOP),D0
000005A0 2140 001E      MOVE.L  D0,BUFSTART(INFOP)      SET NEW BUFFER ADDRESS
                        *                       CALCULATE BLOCKSIZE FOR THIS OPERATION
000005A4 90A8 000E      SUB.L   BBUFFER(INFOP),D0
000005A8 2228 0012      MOVE.L  BCOUNT(INFOP),D1
000005AC 9280           SUB.L   D0,D1           D1 NOW IS BYTES LEFT
```

```
PAGE  19 [3.0] 12/26/84 21:43:09  ASSEMBLY OF BUB_DVR.TEXT                                  *** File name: BUB_DVR ***

 724                          *                                  PAGE SIZE IS ASSUMED TO BE 64 BYTES
 725    00000SAE D2BC 0000     003F          ADD.L   #63,D1  ROUND UP
 726    000005B4 EC81          R2550         ASR.L   #6,D1   DIVIDE BY 64    D1 IS NOW NUMBER OF PAGES
 727    000005B6 0C81 0000     080C          CMPI.L  #MAXPAGES,D1    CLIP IT AT MAXPAGES
 728    000005BC 6D06                        BLT.S   R2586
 729    000005BE 223C 0000     0800          MOVE.L  #MAXPAGES,D1
 730    00000SC4 2141 002A R2586             MOVE.L  D1,BLOCKSIZE(INFOP)
 731
 732    0000C5C8 2F09                        MOVE.L  CARDP,-(SP)     CARD^
 733    000005CA 2F08                        MOVE.L  INFOP,-(SP)     INFO^
 734    000005CC 2F3C 0000     0021          MOVE.L  #$21,-(SP)
 735    000005C2 4EBA FAF0                   JSR     INITIALREGS
 736
 737    000005D6 226E 000C                   MOVEA.L 12(A6),A1
 738    000005DA 2251                        MOVEA.L (A1),CARDP
 739    000005DC 137C 0012     000B          MOVE.B  #READ_DATA,COMMAND(CARDP)
 740    000005E2 6000 00CA                   BRA     ISRDONE
 741
 742                          *              IN B_WRITEING STATE ... HAVE DATA TO WRITE OR END OF OPERATION
 743    000005E6 0812 0007 WRITING BTST      #BUSY,(STATUSP)
 744    000005EA 671A                        BEQ.S   R2716
 745                          *                                  NORMAL WRITE OPERATION
 746    000005EC 2028 0026                   MOVE.L  BUFEND(INFOP),D0        GET END OF BUFFER ADDRESS
 747    000005F0 B08C          R2664         CMP.L   BUFFP,D0        CHECK BUFFER AGAINST BUFFER END
 748    000005F2 670A                        BEQ.S   R2712
 749    000005F4 0812 0000                   BTST    #FIFO_AVAIL,(STATUSP)   CHECK THE FIFO
 750    000005F8 6704                        BEQ.S   R2712
 751    000005FA 169C                        MOVE.B  (BUFFP)+,(DATAP)         WRITE THE DATA
 752    000005FC 60F2                        BRA.S   R2664
 753
 754    000005FE 214C 0022 R2712             MOVE.L  BUFFP,BUFADDR(INFOP)    PUT BACK THE BUFFER ADDRESS
 755    00000602 6000 00AA                   BRA     ISRDONE
 756                          *              NOT BUSY SO OPERATION IS DONE
 757                          *              DID ANYTHING GO WRONG?
 758    00000606 0812 0004 R2716             BTST    #TIME_ERR,(STATUSP)
 759    0000060A 6606                        BNE.S   R2750
 760    0000060C 0812 0005                   BTST    #OP_FAIL,(STATUSP)
 761    00000610 672C                        BEQ.S   R2820
 762                          *                          SOMETHING WENT WRONG SO TRY TO RESTART
 763    00000612 14BC 0020 R2750             MOVE.B  #CLEAR_INT,(COMMANDP)   CLEAR THE INTERUPT
 764    00000616 0829 0006 R2752             BTST    #INT_R,INT_REG(CARDP)   WAIT FOR IT TO GO AWAY
 765    0000061C 66F8          0003          BNE.S   R2752
 766
 767    0000061E 53A8 0016                   SUBQ.L  #1,BRETRY(INFOP)        CHECK THE RETRY COUNTER
 768    00000622 6D0E                        BLT.S   R2806
 769    00000624 2F2E 000C                   MOVE.L  12(A6),-(SP)    TRY AGAIN
 770    00000628 2F2E 0008                   MOVE.L  8(A6),-(SP)
 771    0000062C 4EBA FD4C                   JSR     STARTWRITE
 772    00000630 607C                        BRA.S   ISRDONE
 773                          *              TOO MANY RETRIES
 774    00000632 4268 0006 R2806             CLR.W   RUNSTATE(INFOP) SET RUNSTATE TO B_IDLE
```

```
PAGE  20 [3.0] 12/26/84 21:43:09  ASSEMBLY OF BUB_DVR.TEXT                                  *** File name: BUB_DVR ***

 775    00000636 317C 0008     0008          MOVE.W  #BIOFAIL,ERRORCODE(INFOP)
 776    0000063C 6070                        BRA.S   ISRDONE
 777                          *                          NO ERRORS SO FINISH UP
 778    0000063E 14BC 0020 R2820             MOVE.B  #CLEAR_INT,(COMMANDP)   CLEAR THE INTERUPT
 779    00000642 0829 0006 R2830             BTST    #INT_R,INT_REG(CARDP)   WAIT FOR IT TO GO AWAY
 780    00000648 66F8          0003          BNE.S   R2830
 781
 782    0000064A 2028 000E                   MOVE.L  BBUFFER(INFOP),D0       CALCULATE END OF BUFFER ADDRESS
 783    0000064E D0A8 0012                   ADD.L   BCOUNT(INFOP),D0
 784    00000652 B08C                        CMP.L   BUFFP,D0        ANY MORE TO BE WRITTEN ?
 785    00000654 6754                        BEQ.S   R3008
 786    00000656 2028 002A                   MOVE.L  BLOCKSIZE(INFOP),D0     INCREMENT START PAGE
 787    0000065A D1A8 001A                   ADD.L   D0,BSPAGE(INFOP)
 788
 789    0000065E 2028 0022                   MOVE.L  BUFADDR(INFOP),D0
 790    00000662 2140 001E                   MOVE.L  D0,BUFSTART(INFOP)      SET NEW BUFFER ADDRESS
 791    00000666 2140 0026                   MOVE.L  D0,BUFEND(INFOP)        SAVE BASE FOR BUFEND
 792                          *                          CALCULATE BLOCKSIZE FOR THIS OPERATION
 793    0000066A 90A8 000E                   SUB.L   BBUFFER(INFOP),D0
 794    0000066E 2228 0012                   MOVE.L  BCOUNT(INFOP),D1
 795    00000672 9280                        SUB.L   D0,D1           D1 NOW IS BYTES LEFT
 796                          *                                  PAGE IS ASSUMED TO HAVE 64 BYTES
 797    00000674 EC81          R2926         ASR.L   #6,D1   DIVIDE BY 64    D1 NOW IS PAGES LEFT
 798    00000676 0C81 0000     0800          CMPI.L  #MAXPAGES,D1    CLIP PAGES AT MAXPAGES
 799    0000067C 6D06                        BLT.S   R2962
 800    0000067E 223C 0000     0800          MOVE.L  #MAXPAGES,D1
 801    00000684 2141 002A R2962             MOVE.L  D1,BLOCKSIZE(INFOP)
 802                          *                                  FOR WRITE OPS, BUFEND IS END OF SEGMENT
 803                          *                                  RE-CALCULATE END OF BUFFER
 804    00000688 ED81                        ASL.L   #6,D1           CONVERT BLOCKSIZE TO BYTES (MULTIPLY BY 64)
 805    0000068A D3A8 0026                   ADD.L   D1,BUFEND(INFOP)        ADD IT TO BASE
 806
 807    0000068E 2F09                        MOVE.L  CARDP,-(SP)     CARD^
 808    00000690 2F08                        MOVE.L  INFOP,-(SP)     INFO^
 809    00000692 2F3C 0000     0021          MOVE.L  #$21,-(SP)
 810    00000698 4EBA FA2A                   JSR     INITIALREGS
 811
 812    0000069C 226E 000C                   MOVEA.L 12(A6),A1
 813    000006A0 2251                        MOVEA.L (A1),CARDP
 814    000006A2 137C 0013     000B          MOVE.B  #WRITE_DATA,COMMAND(CARDP)
 815    000006A8 6004                        BRA.S   ISRDONE
 816    000006AA 4268 0006 R3008             CLR.W   RUNSTATE(INFOP) SET RUNSTATE TO B_IDLE
 817    000006AE 4E5E          ISRDONE       UNLK    A6
 818    000006B0 205F                        MOVEA.L (SP)+,A0
 819    000006B2 504F                        ADDQ.W  #8,SP
 820    000006B4 4ED0                        JMP     (A0)
 821
 822              0000 06B6 BUB_DVR_BUB_DVR EQU     *
 823    000006B6 4E75                        RTS
 824
 825                          END
```

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

COMASM

Description

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

Usage

The buffer, interrupt and DMA facilities are used by HP-IB, 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 [3.0] 12/28/84 21:25:39  ASSEMBLY OF COMASM.TEXT                                  *** File name: COMASM ***
PASS 1 COMPLETE. ERRORS: 0
```

```
PAGE   2 [3.0] 12/28/84 21:25:39 IOLIB IOCOMASM - COMMON ASSEMBLY ROUTINES               *** File name: COMASM ***
```

```
                        ********************************************************************************
                        *
                        *       COPYRIGHT (C) 1984 BY HEWLETT-PACKARD COMPANY
                        *
                        ********************************************************************************
                        *
                        *
                        *       IOLIB     IOCOMASM
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *
                        *       Library - IOLIB
                        *       Module  - IOCOMASM
                        *       Author  -
                        *       Phone   -
                        *
                        *       Purpose - This set of assembly language
                        *                 code is intended to be used as
                        *                 a support module for I/O drivers
                        *
                        *       Date    - 08/18/81
                        *       Update  - 08/03/83
                        *       Release - ????????
                        *
                        *
                        *       Source  - IOLIB:COMASM.TEXT
                        *       Object  - IOLIB:COMASM.CODE
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *        NOT RELEASED
                        *       VERSION         3.0
                        *
                        *
                        ********************************************************************************
```

```
PAGE   3 [3.0] 12/26/84 21:25:39 IOLIB IOCOMASM - COMMON ASSEMBLY ROUTINES                *** File name: COMASM ***

                        ********************************************************************************
                        *
                        *       PASCAL DEFINITION OF MODULE
                        *
                        ********************************************************************************
                                MNAME IOCOMASM
                                SRC MODULE IOCOMASM;
                                SRC IMPORT codeclarations;
                                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
```

```
PAGE   4 [3.0] 12/26/84 21:25:39 IOLIB IOCOMASM - COMMON ASSEMBLY ROUTINES                *** File name: COMASM ***

                        ********************************************************************************
                        *
                        *       SYMBOLS FOR EXPORT - COMMON ASSEMBLY LANGUAGE ROUTINES
                        *
                        *       THE SYMBOLS DO NOT HAVE PASCAL ENTRY
                        *       POINTS SINCE THEY ARE ONLY USED BY
                        *       ASSEMBLY LANGUAGE MODULES OR WITH EXTERNAL DECLARATIONS
                        *
                        ********************************************************************************
                                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
                                DEF  TIMEREXISTS    USED AS PASCAL EXTERNAL PROC
                                DEF  TIMED_OUT      USED AS PASCAL EXTERNAL PROC


                        *
                        ********************************************************************************
                        *
                        *          IMPORTED SYMBOLS
                        *
                        ********************************************************************************
                        *
                                REFA   CHECK_TIMER    USED TO GET AT TIMER ROUTINE IN POWERUP
                                LMODE  CHECK_TIMER
```

```
                        *
                        *       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 4E90               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 4E90               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 4E90               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 4E90               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 4E90               JMP     (A0)            return
```

```
170                                 INCLUDE IOLI8:COMDCL
```



```
173                         ********************************************************************************
174                         *
175                         *       modified:  02/22/82  JPC    added parm to user EOT & ISR proc's
176                         *                  08/01/83  JS     added timer_present and sysflag2 equ's
177                         *
178                         ********************************************************************************
179                         *
180                         *       HPL CONVENTIONS
181                         *
182                         *
183                         *         Much of this code is taken intact from the 9826 HPL
184                         *         language system EIO ROM ( extended I/O ROM ).
185                         *
186                         *         The Pascal that will be calling this code uses
187                         *         the stack for parameter passage.  The HPL code
188                         *         uses the Ax and Dx registers for all parameters.
189                         *         The Pascal driver entry points on the previous pages
190                         *         take care of getting the parameters into the correct
191                         *         registers.
192                         *
193                         *
194                         *       GENERAL HPL ENTRY/EXIT CONDITIONS:
195                         *
196                         *         A1.L = CARD ADDRESS
197                         *         A2.L = DRIVER TEMP ADDRESS
198                         *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
199                         *
200                         *
201                         *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
202                         *
203                         *         A3.L = BUFFER CONTROL BLOCK ADDRESS
204                         *                In addition to the A1/A2 convention, Pascal will use
205                         *                A3 for a pointer to the buffer control block.
206                         *                The HPL system keeps much of the transfer
207                         *                information in the s.c. temps.
208                         *
209                         *         TIMEOUT(A2) = contains timeout information
210                         *                Timeout was a global temp in HPL and a timeout
211                         *                generated an error.
212                         *                In PASCAL each card has a timeout value stored in
213                         *                its temporary area.  A timeout error
214                         *                generates an ESCAPE ( which can be trapped ).
215                         *
216                         *
217                         ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *          DRIVER TEMPS TEMPLATE
                        *
                        *            OFFSET FROM A2
                        *
                        *            HPL DECLARATIONS ( MODIFIED )
                        *
                        *
                        ********************************************************************************
000) 0000 ISR_ENTRY     EQU 0       ..19        PASCAL ISR LINK & UNLINK area
000) 0014 USER_ISR      EQU 20                  user ISR:  do NOT change the proc/stat link/parm ordering!!!
000) 0014 H_ISR_PR      EQU 20      ..23            procedure ptr
000) 0018 H_ISR_SL      EQU 24      ..27            static link
000) 001C H_ISR_PM      EQU 28      ..31            parameter
000) 0020 C_ADR         EQU 32      ..35        card address
000) 0024 BUFI_OFF      EQU 36      ..39        buffer pointer offset
000) 0028 BUFO_OFF      EQU 40      ..43        buffer pointer offset
000) 002C EIRB_OFF      EQU 44                  eir byte
000) 002D IO_SC         EQU 45                  select code ( i.e. 7, 22, etc. )
000) 002E TIMEOUT       EQU 46      ..49        timeout value
                        *                           =0 : no timeout
                        *                           #0 : value of timeout
000) 0032 MA_W          EQU 50      ..51        word access to my address
000) 0033 MA            EQU 51                  byte access to my address
000) 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

                0000 0001 TIMER_PRESENT  EQU   1        JS  8/1/83 _SYSFLAG2_BIT -- 0=>TIMER PRESENT
                FFFF FEDA SYSFLAG2       EQU   $FFFFFEDA          JS  8/1/83
```



```
                        ********************************************************************************
                        *
                        *       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 6100 00C6      ABORT_IO3 BSR   ITXFR                     IS THERE A TRANSFER IN PROGRESS?
00000070 672E                   BEQ.S   ABORT_IO2                 IF NOT, DO NOTHING
00000072 B23C 0002              CMP.B   #IT_DMA,D1                ELSE IS IT A DMA?
00000076 671A                   BEQ.S   ABORT_IO1                 IF NOT, SKIP
00000078 6100 021E              BSR     DROPDMA                   ELSE FREE UP THE DMA CH, GET COUNT
0000007C 2744 0010              MOVE.L  D4,TCNT_OFF(A3)           fix up count
00000080 9684                   SUB.L   D4,D3                     fix up actual count
00000082 4A2B 000D              TST.B   TDIR_OFF(A3)
00000086 6606                   BNE.S   AB_OUT
00000088 D7AB 0020              ADD.L   D3,TFIL_OFF(A3)           if input then update fill
0000008C 6004                   BRA.S   ABORT_IO1
0000008E D7AB 001C AB_OUT       ADD.L   D3,TEMP_OFF(A3)           if output then update empty
00000092 177C 00FF ABORT_IO1    MOVE.B  #255,T_SC_OFF(A3)         UNBUSY THE BUFFER
                 0005
00000098 422B 0007              CLR.B   TACT_OFF(A3)              SET TRANSFER TYPE TO NONE
0000009C 4294                   CLR.L   (A4)                      clear buffer ptr
0000009E 600C                   BRA     ABORT_IO3                 see if there is another
000000A0 4E6F           ABORT_IO2 MOVE  (SP)+,SR                  RESTORE USER MODE                  scs
000000A2 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 )
                        *
                        *****************************************************************************
000000A4 4A2B 000D CHECK_TFR TST.B   TDIR_OFF(A3)              base test on direction
000000A8 6608                BNE.S   CHKT_IN                   ( if this is in , check out )
000000AA 49EA 0028 CHKT_OUT  LEA     BUFO_OFF(A2),A4           IS THERE  AN output  BUFFER ACTIVE?
000000AE 2214                MOVE.L  (A4),D1
000000B0 6006                BRA.S   CHKWAIT                   IF SO , THEN WAIT


000000B2 49EA 0024 CHKT_IN   LEA     BUFI_OFF(A2),A4           is there an input  tfr
000000B6 2214                MOVE.L  (A4),D1
                        *    BRA.S   CHKWAIT


000000B8 671C        CHKWAIT BEQ.S   CHKEXIT                   exit if no tfr
000000BA 242A 002E           MOVE.L  TIMEOUT(A2),D2            get timeout value
000000BE 67E4                BEQ.S   CHECK_TFR                 if timeout = 0 then try forever
000000C0 0838 0001           BTST    #TIMER_PRESENT,SYSFLAG2   CHECK IF TIMER PRESENT   JS 8/3/83
                 FEDA
000000C6 6710                BEQ.S   CHKT_TIM                  IF SO THEN USE IT            JS 8/3/83
000000C8 E188                LSL.L   #8,D2
000000CA 2214        CHKLOOP MOVE.L  (A4),D1                   check the buffer again
000000CC 6708                BEQ.S   CHKEXIT                   if finished in time then return
000000CE 5382                SUBQ.L  #1,D2                     decrement
000000D0 66F8                BNE.S   CHKLOOP
000000D2 6000 FF8E           BRA     CTM0_ERR
000000D6 4E75        CHKEXIT RTS


000000D8 1F3C 0001 CHKT_TIM  MOVE.B  #1,-(SP)                  SET UP TIMER RECORD          JS 8/3/83
000000DC 2F02                MOVE.L  D2,-(SP)                                               JS 8/3/83
000000DE 2214        CHKT_TIM1 MOVE.L (A4),D1                  TRANSFER ACTIVE ?            JS 8/3/83
000000E0 670E                BEQ.S   CHKT_TIM2                 NO -- EXIT                   JS 8/3/83
000000E2 4857                PEA     (SP)                      ELSE CHECK TIMER             JS 8/3/83
000000E4 4EB9 0000           JSR     CHECK_TIMER                                            JS 8/3/83
                 0000
000000EA 6AF2                BPL     CHKT_TIM1                 BRANCH IF NOT TIMED OUT      JS 8/3/83
000000EC 6000 FF94           BRA     CTM0_ERR                  ELSE DO TIMEOUT ESCAPE       JS 8/3/83
000000F0 5C8F        CHKT_TIM2 ADDQ.L #6,SP                    CLEAN TIMER RECORD FROM STACK JS 8/3/83
```

```
 459    000000F2 4E75                RTS                          AND RETURN                JS 8/3/83
```



```
 461                         ********************************************************************************
 462                         *
 463                         *       WAIT_TFR
 464                         *
 465                         *         ROUTINE TO CHECK FOR ACTIVE TRANSFER.
 466                         *               ( with a timeout wait )
 467                         *
 468                         *         ENTRY:  A2.L = TEMP POINTER
 469                         *
 470                         *         EXIT :  IF NOT TRANSFER, RETURN
 471                         *                 IF     TRANSFER, THEN wait until finished
 472                         *                                      or   until timeout ( if any )
 473                         *
 474                         *         USES:   NO REGS OTHER THAN RETURN VALUES.
 475                         *
 476                         *
 477                         ********************************************************************************
 478    000000F4 6100 003E WAIT_TFR  BSR    ITXFR                        quick check for tfr
 479    000000F8 671C               BEQ.S  WT_DONE                      and exit
 480    000000FA 2C2A 002E          MOVE.L TIMEOUT(A2),D6               get timeout value
 481    000000FE 67F4               BEQ.S  WAIT_TFR                     if timeout = 0 then try forever
 482    00000100 0838 0001          BTST   #TIMER_PRESENT,SYSFLAG2 IF TIMER PRESENT USE IT   JS 8/3/83
                 FEDA
 483    00000106 6710               BEQ.S  WT_TIM                        BRANCH IF WE HAVE IT      JS 8/3/83
 484    00000108 EB8E               LSL.L  #5,D6
 485    0000010A 6128       WT_LOOP BSR.S  ITXFR                        try
 486    0000010C 6708               BEQ.S  WT_DONE                      if finished in time then return
 487    0000010E 5386               SUBQ.L #1,D6                        decrement
 488    00000110 66F8               BNE.S  WT_LOOP
 489    00000112 6000 FF2E          BRA    CTMO_ERR
 490    00000116 4E75       WT_DONE RTS
 491
 491
 491
 492    00000118 1F3C 0001 WT_TIM   MOVE.B #1,-(SP)                     SET UP TIMER RECORD          JS 8/3/83
 493    0000011C 2F06               MOVE.L D6,-(SP)                                                  JS 8/3/83
 494    0000011E 6114       WT_TIM1 BSR.S  ITXFR                        CHECK FOR ACTIVE TRANSFER    JS 8/3/83
 495    00000120 670E               BEQ.S  WT_TIM2                      NONE -- EXIT                 JS 8/3/83
 496    00000122 4857               PEA    (SP)                         CHECK TIMER                  JS 8/3/83
 497    00000124 4EB9 0000          JSR    CHECK_TIMER                                               JS 8/3/83
                 0000
 498    0000012A 6AF2               BPL    WT_TIM1                      LOOK AGAIN IF NOT TIMED OUT  JS 8/3/83
 499    0000012C 6000 FF14          BRA    CTMO_ERR                     ELSE DO TIMEOUT ESCAPE       JS 8/3/83
 500    00000130 5C8F       WT_TIM2 ADDQ.L #6,SP                        CLEAN UP TIMER RECORD        JS 8/3/83
 501    00000132 4E75               RTS                                 AND RETURN                   JS 8/3/83
```

```
                        ********************************************************************************
                        *
                        *       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 )
                        *
                        ********************************************************************************
00000134 49EA 0024 ITXFR    LEA     BUFI_OFF(A2),A4     IS THERE  AN input  BUFFER ACTIVE?
00000138 2214              MOVE.L  (A4),D1
0000013A 6608              BNE.S   ITXFR3              IF NOT, SKIP
0000013C 49EA 0028         LEA     BUFO_OFF(A2),A4     is there an output tfr
00000140 2214              MOVE.L  (A4),D1
00000142 6722              BEQ.S   ITXFR1                 -no
00000144 2641      ITXFR3  MOVEA.L D1,A3               \
00000146 4281              CLR.L   D1                    ELSE GET BUFFER TYPE WORD
00000148 122B 0007         MOVE.B  TACT_OFF(A3),D1     /
0000014C 4282              CLR.L   D2
0000014E 342B 000E         MOVE.W  TCHR_OFF(A3),D2     GET TERMINATING CHAR
00000152 262B 0010         MOVE.L  TCNT_OFF(A3),D3     GET COUNT
00000156 206B 001C         MOVEA.L TEMP_OFF(A3),A0     GET EMPTY POINTER
0000015A 4A2B 000D         TST.B   TDIR_OFF(A3)        check direction
0000015E 6604              BNE.S   ITXFR2              \ IF INPUT
00000160 206B 0020         MOVEA.L TFIL_OFF(A3),A0     /    THEN GET FILL POINTER
00000164 7A01      ITXFR2  MOVEQ   #1,D5               set not zero
                   0000 0166 STCLR1  EQU     *
00000166 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 )
                        *
                        ********************************************************************************
00000168 61CE      STCLR   BSR     ITXFR               GET BUFFER POINTER FROM TEMPS
0000016A 67FA              BEQ.S   STCLR1              IF ALREADY CLEAR, SKIP
0000016C 177C 00FF         MOVE.B  #255,T_SC_OFF(A3)   CLEAR S.C. INDICATOR IN THE BUF CTL BLK
                   0005
00000172 422B 0007         CLR.B   TAC_OFF(A3)         clear tfr type
00000176 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
                        *
                        ********************************************************************************
00000178 41EB 0024 LOGEOT   LEA     T_PR_OFF(A3),A0     point to procedure/static link/parameter

0000017C 2010      H_EOT1  MOVE.L  (A0),D0             is there a proc?
0000017E 671A              BEQ.S   H_EOT3              skip if not

00000180 48E7 0078         MOVEM.L A1-A4,-(SP)         save dedicated regs      (8/10/82 JPC)
00000184 2F28 0008         MOVE.L  8(A0),-(SP)         push the parameter
00000188 2228 0004         MOVE.L  4(A0),D1            is there a static link?
0000018C 6702              BEQ.S   H_EOT2              ---------- says it is okay to try
0000018E 2F01              MOVE.L  D1,-(SP)                 and call proc with static link
00000190 2040      H_EOT2  MOVEA.L D0,A0               procedure address
00000192 4E90              JSR     (A0)                call it
00000194 4CDF 1E00         MOVEM.L (SP)+,A1-A4         restore dedicated regs   (8/10/82 JPC)

00000198 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
                        *
                        ********************************************************************************
0000019A 41EA 0014 LOGINT   LEA     H_ISR_PR(A2),A0       point to procedure/static link/parameter
0000019E 60DC               BRA     H_EOTI                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 )
                        *
                        ********************************************************************************
000001A0 214C 0000 DMA_STBSY MOVE.L A4,DMAISR(A0)          SAVE THE TERMINATION ROUTINE
000001A4 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 )
                        *
                        ********************************************************************************
000001A8 2740 0010 STBSY    MOVE.L D0,TCNT_OFF(A3)        COPY TFR COUNT INTO TEMPS.
000001AC 4A2B 000D          TST.B  IDIR_OFF(A3)           \
000001B0 6606               BNE.S  STBSY1                  \
000001B2 2548 0024          MOVE.L A3,BUFI_OFF(A2)          MAKE SELECT CODE BUSY
000001B6 6004               BRA.S  STBSY2                  /
000001B8 2548 0028 STBSY1   MOVE.L A3,BUFO_OFF(A2)        /
000001BC 176A 002D STBSY2   MOVE.B IO_SC(A2),T_SC_OFF(A3) SET UP BUFFER ACTIVE SELECT CODE
           0005
000001C2 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 01C4 IOCOMASM_DMA_REQUEST EQU *
000001C4 285F                   MOVEA.L (SP)+,A4       save return address
000001C6 245F                   MOVEA.L (SP)+,A2       get sc temp
000001C8 226A 0020              MOVEA.L C_ADR(A2),A1   get card ptr
000001CC 4E4B                   TRAP  #11              GET INTO SUPERVISOR MODE        scs
                        * scs   MOVE    SR,-(SP)       JUST IN CASE CALLER DIDN'T DISABLE
000001CE 007C 2700              ORI     #$2700,SR      INTERRUPTS, I WILL.
000001D2 6100 0080              BSR     TESTDMA        SEE IF DMA IS INSTALLED
000001D6 6710                   BEQ.S   DR_FAIL        IF NOT, return -1
000001D8 5343                   SUBQ.W  #1,D3          turn $82/$81 to $81/$80
000001DA 0243 0001              ANDI.W  #1,D3          determine channel
000001DE 4803                   EXT.L   D3
000001E0 116A 002D              MOVE.B  IO_SC(A2),DMA_SC(A0)    ELSE CLAIM THIS CHANNEL FOR CALLER
         FFFF
000001E6 6006                   BRA.S   DR_GOOD
000001E8 263C FFFF DR_FAIL      MOVE.L  #-1,D3         return -1
         FFFF
000001EE 46DF           DR_GOOD MOVE    (SP)+,SR       restore int. state
000001F0 2E83                   MOVE.L  D3,(SP)        assign return value - channel ( or -1 )
000001F2 4ED4                   JMP     (A4)           return addr


                        *
                        *       release a dma channel
                        *
                0000 01F4 IOCOMASM_DMA_RELEASE EQU *
000001F4 205F                   MOVEA.L (SP)+,A0       save return address
000001F6 245F                   MOVEA.L (SP)+,A2       get sc temp
000001F8 226A 0020              MOVEA.L C_ADR(A2),A1   get card ptr
000001FC 4850                   PEA     (A0)           push return address
000001FE 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 )
                        *
                        ********************************************************************************
00000202 4E4B           GETDMA    TRAP  #11                     GET INTO SUPERVISOR MODE        scs
                        * scs     MOVE  SR,-(SP)                JUST IN CASE CALLER DIDN'T DISABLE
00000204 007C 2700              ORI   #$2700,SR               INTERRUPTS, I WILL.
00000208 0C80 0001              CMPI.L #$010001,D0            \ make sure count <=65536
         0001
0000020E 6A00 FE36              BPL   TERR_C                  /
00000212 6140                   BSR.S TESTDMA                 SEE IF DMA IS INSTALLED
00000214 6700 FE38              BEQ   TERR_D                  IF NOT, GIVE ERROR
00000218 116A 002D              MOVE.B IO_SC(A2),DMA_SC(A0)      ELSE CLAIM THIS CHANNEL FOR CALLER
         FFFF
0000021E 2842                   MOVEA.L D2,A4                 A4 = ADDRESS OF DMA CHANNEL HARDWARE.

00000220 242B 001C              MOVE.L TEMP_OFF(A3),D2        \
00000224 4A2B 0000              TST.B  TDIR_OFF(A3)           \
00000228 6604                   BNE.S  GETDMA1                   SET UP ADDRESS
0000022A 242B 0020              MOVE.L TAIL_OFF(A3),D2        /
0000022E 28C2           GETDMA1   MOVE.L D2,(A4)+             /

00000230 5380                   SUBQ.L #1,D0                     COUNT REGISTERS (COUNT REG
00000232 38C0                   MOVE.W D0,(A4)+                  MUST BE COUNT-1)
00000234 5280                   ADDQ.L #1,D0


00000236 4242                   CLR.W  D2
00000238 142B 000D              MOVE.B TDIR_OFF(A3),D2        MOVE DIRECTION BIT INTO B2 OF D2
0000023C E54A                   LSL    #2,D2                  IN ORDER TO CONSTRUCT DMA ARM


0000023E 4A2B 000A              TST.B  T_BW_OFF(A3)           IF BYTE TRANSFER
00000242 6702                   BEQ.S  GETDMA2                THEN SKIP
```



```
00000244 5442                   ADDQ.W #2,D2                  ELSE SET BIT 1 OF DMA ARM.
00000246 4A2B 0030 GETDMA2      TST.B  T_DMAPRI(A3)           check for dma priority requested
0000024A 6702                   BEQ.S  GETDMA3
0000024C 5042                   ADDQ.W #8,D2                    if set  then set pri bit
0000024E 5242           GETDMA3   ADDQ.W #1,D2                SET BITO OF DMA ARM
00000250 46DF                   MOVE  (SP)+,SR                                        scs
00000252 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 )
                        *
                        ********************************************************************************
00000254 41ED FFC3 TESTDMA    LEA    DMAFLAG(A5),A0         \
00000258 4A10                 TST.B  (A0)                   \ DO RET 1 IF NO DMA
0000025A 673A                 BEQ.S  TESTDMA_C              /
0000025C B3FC 0047            CMPA.L #H_INT_CA,A1           IF THIS IS A REQUEST FOR THE INTERNAL
         8000
00000262 6716                 BEQ.S  TESTDMA_A              HP-IB, THEN CAN'T TRY FOR CH 1 SO SKIP.
00000264 41ED FFEE            LEA    DMA1(A5),A0            ELSE ASSUME WE CAN GET CH 1
00000268 243C 0050            MOVE.L #DMACH1,D2
         0008
                        * tm    MOVEQ  #$82,D3
0000026E 163C 0082            MOVE.B #$82,D3
00000272 0C28 00FF            CMPI.B #255,DMA_SC(A0)        CAN WE?
         FFFF
00000278 671A                 BEQ.S  TESTDMA_B              IF SO, THEN RET 3
0000027A 41ED FFF8 TESTDMA_A  LEA    DMA0(A5),A0            ELSE ASSUME WE CAN GET CH 0
0000027E 243C 0050            MOVE.L #DMACH0,D2
         0000
                        * tm    MOVEQ  #$81,D3
00000284 163C 0081            MOVE.B #$81,D3
00000288 0C28 00FF            CMPI.B #255,DMA_SC(A0)        CAN WE?
         FFFF
0000028E 6704                 BEQ.S  TESTDMA_B              IF HARDWARE PRESENT BUT BUSY,same as not there
00000290 4245                 CLR    D5
00000292 4E75                 RTS
00000294 7A01           TESTDMA_B MOVEQ  #1,D5              ELSE WE GOT A CH
```

```
PAGE  25 [3.0] 12/.6/84 21:25:39 IOLIB IOCOMASM - ASSEMBLY DMA PROCEDURES                *** File name: COMASM ***

  831    00000298 4E75      TESTDMA_C RTS
  832
  832
  832
  832
  832
  832
  833                       ********************************************************************************
  834                       *
  835                       *       DROPDMA
  836                       *
  837                       *         ROUTINE TO FREE UP A DMA CHANNEL
  838                       *
  839                       *         ENTRY:  A2.L = POINTER TO DRIVER TEMPS
  840                       *
  841                       *         EXIT:   D4.W = FINAL DMA CHANNEL COUNT
  842                       *                 CHANNEL IS DISARMED.
  843                       *
  844                       *         USES:   A0
  845                       *
  846                       *         HPL ROUTINE ( MODIFIED )
  847                       *
  848                       ********************************************************************************
  849             0000 0298 DROPDMA   EQU *
  850    00000298 4E4B                TRAP #11                                                    scs
  851                       * scs     MOVE   SR,-(SP)             JUST IN CASE CALLER DIDN'T DISABLE
  852    0000029A 007C 2700           ORI    #$2700,SR            INTERRUPTS, I WILL.
  853    0000029E 7800                MOVEQ  #0,D4                ASSUME DMA CHA ALREADY DROPPED...
  854    000002A0 102D FFF7           MOVE.B DMA_SC_0(A5),D0      \
  855    000002A4 B02A 002D           CMP.B  IO_SC(A2),D0         / IS IT CH 0?
  856    000002A8 660C                BNE.S  DROPDMA0             IF NOT, SKIP
  857    000002AA 49ED FFF8           LEA    DMA0(A5),A4          GET A POINTER TO THE CHANNEL R/W
  858    000002AE 41F9 0050           LEA    DMACH0,A0            POINT A0 TO CH 0
                0000
  859    000002B4 6014                BRA.S  DROPDMA1             GO DO IT
  860
  860
  861    000002B6 102D FFED DROPDMA0  MOVE.B DMA_SC_1(A5),D0      \
  862    000002BA B02A 002D           CMP.B  IO_SC(A2),D0         / IS IT CH 1?
  863    000002BE 6622                BNE.S  DROPDMA2             IF NOT, DO NOTHING
  864    000002C0 49ED FFEE           LEA    DMA1(A5),A4          GET A POINTER TO THE CHANNEL R/W
  865    000002C4 41F9 0050           LEA    DMACH1,A0            POINT A0 TO CH 1
                0008
  866    000002CA 197C 00FF DROPDMA1  MOVE.B #255,DMA_SC(A4)      clear s.c.
                FFFF
  867    000002D0 42AC 0004           CLR.L  DMASL(A4)            clear static link
  868    000002D4 42AC 0000           CLR.L  DMAISR(A4)           clear isr pointer
  869    000002D8 2818                MOVE.L (A0)+,D4             DISARM CH BY READING ADDRESS
  870    000002DA 4284                CLR.L  D4
  871    000002DC 3810                MOVE.W (A0),D4              GET FINAL COUNT INTO D0
  872    000002DE D87C 0001           ADD.W  #1,D4                FIX UP COUNT TO INDICATE LEFT OVER TFR'S
  873    000002E2 46DF      DROPDMA2  MOVE   (SP)+,SR                                             scs
  874    000002E4 4E75                RTS                                                         scs
  875                       * scs     RTE
```

```
PAGE  26 [3.0] 12/26/84 21:25:39 IOLIB IOCOMASM - ASSEMBLY DMA PROCEDURES                *** File name: COMASM ***

  877                       *
  878                       ********************************************************************************
  879                       *
  880                       *         TIMEREXISTS:  PASCAL FUNCTION TO SEE IF TIMER EXISTS
  881                       *
  882                       *           FUNCTION TIMEREXISTS: BOOLEAN; EXTERNAL;
  883                       *
  884                       *           RETURNS TRUE IF TIMER PRESENT, ELSE FALSE
  885                       *
  886                       *           --------- 8/2/83
  887                       *
  888                       ********************************************************************************
  889                       *
  890             0000 02E6 TIMEREXISTS EQU *
  891    000002E6 0838 0001           BTST   #TIMER_PRESENT,SYSFLAG2  CHECK BIT FOR TIMER PRESENT
                FEDA
  892    000002EC 57EF 0004           SEQ    4(SP)                  SET FUNCTION RESULT 0=>TRUE
  893    000002F0 4E75                RTS                             AND RETURN
  894                       *
  895                       ********************************************** **************************
  896                       *
  897                       *         TIMED_OUT:   PASCAL FUNCTION TO SEE IF TIMEOUT HAS OCCURRED
  898                       *
  899                       *            FUNCTION TIMED_OUT(VAR REC: TIMEOUTREC): BOOLEAN; EXTERNAL;
  900                       *
  901                       *            TIMEOUTREC= PACKED RECORD
  902                       *                           COUNT: INTEGER ( SET TO TIMEOUT IN MS )
  903                       *                           FIRSTTIME: BOOLEAN; {SET THIS TO TRUE FOR FIRST
  904                       *                                                CALL )
  905                       *                         END;
  906                       *
  907                       *            RETURNS:  TRUE IF TIMEOUT PERIOD EXPIRED, ELSE FALSE
  908                       *
  909                       *            CAUTION: WILL SMASH BOTH PARTS OF TIMEOUTREC PARAMETER
  910                       *
  911                       *                --------- 8/2/83
  912                       *
  913                       *                  REMOVED ADDQ.L #4,SP AFTER SMI   JWS 5/3/84
  914                       ********************************************************************************
  915                       *
  916    000002F2 205F      TIMED_OUT      MOVEA.L   (SP)+,A0     SAVE RETURN ADDRESS
  917    000002F4 4EB9 0000                JSR       CHECK_TIMER  CALL CHECK_TIMER USING PARAMETER ON STK
                0000
  918    000002FA 58D7                     SMI       (SP)         SET RESULT OF FUNCTION
  919    000002FC 4ED0                     JMP       (A0)          AND RETURN WITH SP POINTING TO RESULT
  920
  921
  922
  923                              END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
CHECK_TIMER	ABS	106	00000002
IODECLARATIONS	ABS	305	00000005
SYSGLOBALS	ABS	306	00000009

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	386		00000066
ABORT_IO1	REL	401		00000092
ABORT_IO2	REL	405		000000A0
ABORT_IO3	REL	389		0000006C
AB_OUT	REL	400		0000008E
AVAIL_OFF	ABS	245		00000034
BADTMO	ABS	323		0000000B
BAD_RDS	ABS	331		00000013
BAD_SCT	ABS	332		00000014
BITT_EXIT	REL	126		00000010
BUFI_OFF	ABS	236		00000024
BUFO_OFF	ABS	237		00000028
BUF_BUSY	ABS	321		00000009
CCR	STREG	0		00000005
CHECK_TFR	REL	427		000000A4
CHKEXIT	REL	448		000000D6
CHKLOOP	REL	443		000000CA
CHKT_IN	REL	433		000000B2
CHKT_OUT	REL	429		000000AA
CHKT_TIM	REL	450		000000D8
CHKT_TIM1	REL	452		000000DE
CHKT_TIM2	REL	458		000000F0
CHKWAIT	REL	437		000000B8
CRD_DWN	ABS	333		00000015
CTMO_ERR	REL	355		00000042
C_ADR	ABS	235		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
DFC	STREG	0		00000008
DMA0	ABS	673	IODECLARATIONS +	FFFFFFF8

DMA1	ABS	678	IODECLARATIONS +	FFFFFFEE
DMACH0	ABS	687		00500000
DMACH1	ABS	688		00500008
DMAFLAG	ABS	671	IODECLARATIONS +	FFFFFFC3
DMAISR	ABS	683		00000000
DMAISR_0	ABS	674	IODECLARATIONS +	FFFFFFF8
DMAISR_1	ABS	679	IODECLARATIONS +	FFFFFFEE
DMASL	ABS	684		00000004
DMASL_0	ABS	675	IODECLARATIONS +	FFFFFFFC
DMASL_1	ABS	680	IODECLARATIONS +	FFFFFFF2
DMA_SC	ABS	685		FFFFFFFF
DMA_SC_0	ABS	676	IODECLARATIONS +	FFFFFFF7
DMA_SC_1	ABS	681	IODECLARATIONS +	FFFFFFED
DMA_STBSY	REL	625		000001A0
DROPDMA	REL	849		00000298
DROPDMA0	REL	861		000002B6
DROPDMA1	REL	866		000002CA
DROPDMA2	REL	873		000002E2
DR_FAIL	REL	714		000001E8
DR_GOOD	REL	715		000001EE
EIRB_OFF	ABS	238		0000002C
EOD_SEEN	ABS	334		00000016
ESC_CODE	ABS	342	SYSGLOBALS +	FFFFFFFE
ESC_ERR	REL	364		00000050
GETDMA	REL	758		00000202
GETDMA1	REL	772		0000022E
GETDMA2	REL	785		00000246
GETDMA3	REL	788		0000024E
H_EOT1	REL	577		0000017C
H_EOT2	REL	585		00000190
H_EOT3	REL	589		00000198
H_INT_CA	ABS	694		00478000
H_ISR_PM	ABS	234		0000001C
H_ISR_PR	ABS	232		00000014
H_ISR_SL	ABS	233		00000018
IOCOMASM_BINAND	REL	131		00000014
IOCOMASM_BINCMP	REL	161		00000038
IOCOMASM_BINEOR	REL	151		0000002C
IOCOMASM_BINIOR	REL	141		00000020
IOCOMASM_BIT_SET	REL	118		00000002
IOCOMASM_DMA_RELEASE	REL	722		000001F4
IOCOMASM_DMA_REQUEST	REL	700		000001C4
IOCOMASM_IOCOMASM	REL	113		00000000
IOE_ERROR	ABS	337		FFFFFFE6
IOE_RSLT	ABS	339	IODECLARATIONS +	FFFFFFBE
IOE_SC	ABS	340	IODECLARATIONS +	FFFFFFBA
IO_MISC	ABS	335		00000017
IO_SC	ABS	239		0000002D
ISR_ENTRY	ABS	230		00000000
ITXFR	REL	522		00000134
ITXFR1	REL	540		00000166
ITXFR2	REL	538		00000164
ITXFR3	REL	528		00000144
LOGEOT	REL	575		00000178
LOGINT	REL	605		0000019A
MA	ABS	244		00000033
MA_W	ABS	243		00000032

NOT_HPIB	ABS	314		00000002
NOT_LSTN	ABS	328		00000010
NOT_TALK	ABS	327		0000000F
NO_ACTL	ABS	315		00000003
NO_CARD	ABS	313		00000001
NO_DATA	ABS	318		00000006
NO_DMA	ABS	325		0000000D
NO_DRV	ABS	324		0000000C
NO_DVC	ABS	316		00000004
NO_SCTL	ABS	330		00000012
NO_SPACE	ABS	317		00000005
NO_WORD	ABS	326		0000000E
RCVR_BLK	ABS	343	SYSGLOBALS +	FFFFFFF6
SC_BUSY	ABS	320		00000008
SFC	STREG	0		00000009
SP	AREG	0		00000007
SR	STREG	0		00000006
STBSY	REL	644		000001A8
STBSY1	REL	649		000001B8
STBSY2	REL	650		000001BC
STCLR	REL	557		00000168
STCLR1	REL	539		00000166
SYSFLAG	ABS	346		FFFFFEDA
TACT_OFF	ABS	260		00000007
TBSZ_OFF	ABS	283		00000018
TBUF_OFF	ABS	282		00000014
TCHR_OFF	ABS	279		0000000E
TCNTERR	ABS	322		0000000A
TCNT_OFF	ABS	281		00000010
TDIR_OFF	ABS	277		0000000D
TEMP_OFF	ABS	284		0000001C
TEND_OFF	ABS	275		0000000B
TERR_B	REL	359		0000004A
TERR_C	REL	367		00000046
TERR_D	REL	361		0000004E
TESTDMA	REL	811		00000254
TESTDMA_A	REL	822		0000027A
TESTDMA_B	REL	830		00000294
TESTDMA_C	REL	831		00000296
TFIL_OFF	ABS	285		00000020
TFR_ERR	ABS	319		00000007
TIMED_OUT	REL	916		000002F2
TIMEOUT	ABS	240		0000002E
TIMEREXISTS	REL	890		000002E6
TIMER_PRESENT	ABS	345		00000001
TMO_ERR	ABS	329		00000011
TTMP_OFF	ABS	258		00000000
TT_BURST	ABS	296		00000003
TT_DMA	ABS	295		00000002
TT_FHS	ABS	297		00000004
TT_INT	ABS	294		00000001
TUSR_OFF	ABS	261		00000009
T_BW_OFF	ABS	273		0000000A
T_DMAPRI	ABS	290		00000030
T_PM_OFF	ABS	289		0000002C
T_PR_OFF	ABS	286		00000024
T_SC_OFF	ABS	259		00000005

T_SL_OFF	ABS	288		00000028
USER_ISR	ABS	231		00000014
USP	STREG	0		00000007
VBR	STREG	0		0000000A
WAIT_TFR	REL	478		000000F4
WT_DONE	REL	490		00000116
WT_LOOP	REL	485		0000010A
WT_TIM	REL	492		00000118
WT_TIM1	REL	494		0000011E
WT_TIM2	REL	500		00000130

Description

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

Usage

DC is used for the 98628 and 98629 interfaces.

Requirements

DC_DRV

```
PAGE   1 [3.0] 12/26/84 21:32:35  ASSEMBLY OF DC.TEXT                                  *** File name: DC ***
PASS 1 COMPLETE. ERRORS: 0
```

```
PAGE   2 [3.0] 12/26/84 21:32:35 IOLIB EXTDC - DATA COMM DRIVERS                    *** File name: DC ***

    3                       ********************************************************************************
    4                       *
    5                       *       COPYRIGHT (C) 1984 BY HEWLETT-PACKARD COMPANY
    6                       *
    7                       ********************************************************************************
    8                       *
    9                       *
   10                       *       IOLIB     EXTDC
   11                       *
   12                       *
   13                       ********************************************************************************
   14                       *
   15                       *
   16                       *
   17                       *       Library - IOLIB
   18                       *       Authors -
   19                       *       Phone   -
   20                       *
   21                       *       Purpose - This set of assembly language code is intended to be used as
   22                       *                 a PASCAL module for I/O drivers for use by the external I/O
   23                       *                 procedures library.
   24                       *
   25                       *
   26                       *
   27                       *                 The code was taken almost directly from the 9826 BASIC data
   28                       *                 comm code.
   29                       *
   30                       *       Date    - 10/26/81
   31                       *       Update  - 05/03/84
   32                       *       Release - 5/15/84
   33                       *
   34                       *
   35                       *       Source  - IOLIB:DC.TEXT
   36                       *       Object  - IOLIB:DC.CODE
   37                       *
   38                       *
   39                       ********************************************************************************
   40                       *
   41                       *
   42                       *       RELEASED
   43                       *       VERSION         3.0
   44                       *
   45                       *
   46                       ********************************************************************************
```

```
*****************************************************************************
*                                                                           *
*                                                                           *
*      BUG FIX HISTORY          - after release 1.0                         *
*                                                                           *
*                                                                           *
*      BUG #   BY  / ON         LOC              DESCRIPTION               *
*      -----   ------------     --------------   ----------------------    *
*                                                                           *
*      1249    ------------     file DC_INTER    tfr count is off by one   *
*              01/08/82         inxex1           byte for overlap tfrs.    *
*                                                                           *
*      SPRyyy  ------------     file DC_COMM     missing instruction in the*
*              06/15/82         alvinit          reset code - not a problem*
*                                                for current use - but for *
*                                                future smart card intrfcs.*
*                                                                          *
*      185     ------------     file DC_INTER    eot and user isr's were   *
*              07/30/82         try_hock         not getting a parameter.  *
*                               try_hock_P                                 *
*                               inxdn1                                     *
*                               outxdn1                                    *
*                                                                          *
*      475     ------------     no where         Change BSRs into JSRs to  *
*              09/17/82                          allow re-placement of the*
*                                                modules.  Also in GPIO and*
*                                                HPIB.  No refs were used  *
*                                                in the data comm modules. *
*                                                                          *
*      tttt    ---------        gain_access      Timing/exception fixes for*
*              08/11/83         direct_control   680xx processors.         *
*              05/03/84         direct_command                             *
*                               wait_outxfrdone                            *
*                               output_data                                *
*                               putctrlblk                                 *
*                               release_access                             *
*                                                                          *
*****************************************************************************
```



```
*************************************************
*                                               *
*                                               *
*       ****     ***    *****    ***           *
*       *   *   *   *     *     *   *          *
*       *   *   *   *     *     *   *          *
*       *   *   *****     *     *****          *
*       *   *   *   *     *     *   *          *
*       *   *   *   *     *     *   *          *
*       ****    *   *     *     *   *          *
*                                               *
*                                               *
*        ***     ***    *   *    *   *          *
*       *   *   *   *   ** **   ** **          *
*       *       *   *   * * *   * * *          *
*       *       *   *   *   *   *   *          *
*       *       *   *   *   *   *   *          *
*       *   *   *   *   *   *   *   *          *
*        ***     ***    *   *   *   *          *
*                                               *
*                                               *
*************************************************
*
*       dc_doc
*       comdcl
*       dc_decls
*       dc_buff
*       dc_comm
*       dc_inter
*       dc_rxbuf
*       dc_trans
*       dc_txbuf

    sprint
    llen    132
```

```
123                     ********************************************************************************
124                     *
125                     *
126                     *       The following lines are used to tell the LINKER/LOADER what this module
127                     *       looks like in PASCAL terms.
128                     *
129                     *       Note that it is possible to create assembly modules that are functions.
130                     *       These routines are called through an indirect pointer using the CALL
131                     *       facility which does NOT permit functions.
132                     *
133                     *       This module is called 'EXTDC' ( upper or lower case - doesn't matter )
134                     *       independent of the file name ( by use of the MNAME pseudo-op ).
135                     *
136                     *       All the externally used procedures are called 'EXTDC_@@@@@@@@' in
137                     *       this module.  If you are using assembly to access them use the
138                     *       'EXTDC_@@@@@@@' name.  If you are using Pascal use the '@@@@@@@'
139                     *       name.
140                     *
141                     ********************************************************************************
142                             MNAME EXTDC
143                             SRC MODULE EXTDC;
144                             SRC IMPORT iodeclarations;
145                             SRC EXPORT
146                             SRC
147                             SRC         PROCEDURE alvinit        ( temp : ANYPTR );
148                             SRC         PROCEDURE alvinisr       ( temp : PISRIB );
149                             SRC         PROCEDURE enter_data     ( temp : ANYPTR ;  x     : ANYPTR ;
150                             SRC                                                     VAR c : INTEGER );
151                             SRC         PROCEDURE output_data    ( temp : ANYPTR ;  x     : ANYPTR ;
152                             SRC                                                     cnt   : INTEGER );
153                             SRC         PROCEDURE output_end     ( temp : ANYPTR );
154                             SRC         PROCEDURE direct_status  ( temp : ANYPTR ;  reg   : io_word;
155                             SRC                                                     VAR x : io_word);
156                             SRC         PROCEDURE direct_control ( temp : ANYPTR ;  reg   : io_word;
157                             SRC                                                     val   : io_word );
158                             SRC         PROCEDURE control_bfd    ( temp : ANYPTR ;  reg   : io_word;
159                             SRC                                                     val   : io_word );
160                             SRC         PROCEDURE start_tfr_out  ( temp : ANYPTR );
161                             SRC         PROCEDURE start_tfr_in   ( temp : ANYPTR );
162                             SRC END; { of extdc }
```



```
164                     ********************************************************************************
165                     *
166                     *       SYMBOLS FOR EXPORT AS PROCEDURE NAMES
167                     *
168                     ********************************************************************************
169                             DEF EXTDC_EXTDC
170
171                             DEF EXTDC_ALVINIT
172                             DEF EXTDC_ALVINISR
173                             DEF EXTDC_ENTER_DATA
174                             DEF EXTDC_OUTPUT_DATA
175                             DEF EXTDC_OUTPUT_END
176                             DEF EXTDC_DIRECT_STATUS
177                             DEF EXTDC_DIRECT_CONTROL
178                             DEF EXTDC_CONTROL_BFD
179                             DEF EXTDC_START_TFR_IN
180                             DEF EXTDC_START_TFR_OUT
181
182                     ********************************************************************************
183                     *
184                     *       SYMBOLS FOR IMPORT - not used currently in data comm
185                     *                            EXCEPT FOR CHECK_TIMER,M68KTYPE   tttt JS 8/11/83
186                     *                            if they are ever used - use a JSR to call them
187                     *
188                     ********************************************************************************
189                     * REFA STBSY
190                     * REFA STCLR
191                     * REFA ITXFR
192                     * REFA ABORT_IO
193                     * REFA LOGINT
194                     * REFA GETDMA
195                     * REFA DROPDMA
196                     * REFA TESTDMA
197                     * REFA DMA_STBSY
198                       REFA CHECK_TIMER                                          tttt JS 8/11/83
199                       REFA M68KTYPE
200                     *
201                     *       change references to allow long jumps when the I/O      475 TM 9/17/82
202                     *       modules get moved                                       475 TM 9/17/82
203                     * LMODE STBSY
204                     * LMODE STCLR
205                     * LMODE ITXFR
206                     * LMODE ABORT_IO
207                     * LMODE LOGINT
208                     * LMODE GETDMA
209                     * LMODE DROPDMA
210                     * LMODE TESTDMA
211                     * LMODE DMA_STBSY
212                       LMODE CHECK_TIMER                                         tttt JS 8/11/83
```

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


                        *
                        *         MODULE initialization
                        *
            0000 0000 EXTDC_EXTDC             EQU *
00000000 4E75                                 RTS
                        *
                        *         ENTRY POINTS
                        *
00000002 6000 0112 EXTDC_ALVINIT              BRA ALVINIT
00000006 6000 0372 EXTDC_ALVINISR             BRA TOP_ISR
0000000A 6000 0910 EXTDC_ENTER_DATA           BRA ENTER_DATA
0000000E 6000 087C EXTDC_OUTPUT_DATA          BRA OUTPUT_DATA
00000012 6000 09FE EXTDC_OUTPUT_END           BRA OUTPUT_END
00000016 6000 01C8 EXTDC_DIRECT_STATUS        BRA DIRECT_STATUS
0000001A 6000 0144 EXTDC_DIRECT_CONTROL       BRA DIRECT_CONTROL
0000001E 6000 0A78 EXTDC_CONTROL_BFD          BRA CONTROL_BFD
00000022 6000 0680 EXTDC_START_TFR_IN         BRA START_TRANSFER_IN
00000026 6000 06A2 EXTDC_START_TFR_OUT        BRA START_TRANSFER_OUT
```



```
                        ************************************************
                        *                                              *
                        *   The following escapes are possible:        *
                        *                                              *
                        *       EOD_SEEN - on ENTER_DATA only,         *
                        *          signifies termination with control  *
                        *          block when too few bytes have been  *
                        *          read.                               *
                        *                                              *
                        *       TMO_ERR - on ENTER_DATA if Rx queue    *
                        *          was empty for too long              *
                        *               - on OUTPUT_DATA if Tx queue   *
                        *          goto blocked up for too long        *
                        *               - on CONTROL_BFD if Tx control *
                        *          queue got blocked up for too long   *
                        *               - on OUTPUT_END if Tx control  *
                        *          queue got blocked up for too long   *
                        *                                              *
                        *                                              *
                        *       CRD_DWN - on every routine if card     *
                        *          was totally locked up for more than *
                        *          one second.                         *
                        *                                              *
                        *   There are no checks for nested I/O.        *
                        *                                              *
                        ************************************************


                        * module: DC_COMM
                        ************************************************
                        *                                              *
                        *   procedure DIRECT_CONTROL (                 *
                        *               var SCT:  select_code_table;   *
                        *               REG:      1..127;              *
                        *               VAL:      0..255  );           *
                        *                                              *
                        *   The ranges of REG & VAL are not checked    *
                        *   for validity.                              *
                        *                                              *
                        ************************************************
                                nosyms
```

```
290                     * module: DC_COMM
291                     ************************************************
292                     *                                              *
293                     *   procedure DIRECT_STATUS (                  *
294                     *               var SCT:  select_code_table;   *
295                     *               REG:      0..127;              *
296                     *               var VAL:  word    );           *
297                     *                                              *
298                     *   These registers are intercepted:           *
299                     *       0:  Gives value from RESET_ID          *
300                     *       1:  Returns true if hardware interrupts *
301                     *       2:  Returns 0                          *
302                     *       5:  Returns 2 bits saying state of Rx  *
303                     *           buffer                             *
304                     *       9:  Returns last ENTER TERM            *
305                     *       10: Returns last ENTER MODE            *
306                     *       11: Returns # bytes available in Tx    *
307                     *           queue, or 0 if there's not 3       *
308                     *           control block positions available  *
309                     *                                              *
310                     *   The range of REG is not checked for        *
311                     *   validity.                                  *
312                     *                                              *
313                     ************************************************
314
315
316
317
318
319
320
321
322
323
324                     * module: DC_TRANS
325                     ************************************************
326                     *                                              *
327                     *   procedure OUTPUT_DATA (                    *
328                     *               var SCT:  select_code_table;   *
329                     *               PTR:      ^ data_bytes;        *
330                     *               COUNT:    longword       );    *
331                     *                                              *
332                     *   This operation may hang waiting for space. *
333                     *                                              *
334                     ************************************************
```



```
336                     * module: DC_INTER
337                     ************************************************
338                     *                                              *
339                     *   procedure START_TRANSFER_IN (              *
340                     *               var SCT: select_code_table );  *
341                     *                                              *
342                     *       This starts the card doing a transfer. *
343                     *       The calling code must have already     *
344                     *       linked the transfer block in to the    *
345                     *       select_code_table structure.           *
346                     *                                              *
347                     ************************************************
348
349
350
351
352
353
354
355                     * module: DC_INTER
356                     ************************************************
357                     *                                              *
358                     *   procedure START_TRANSFER_OUT (             *
359                     *               var SCT: select_code_table );  *
360                     *                                              *
361                     *       This starts the card doing a transfer. *
362                     *       The calling code must have already     *
363                     *       linked the transfer block in to the    *
364                     *       select_code_table structure.           *
365                     *                                              *
366                     ************************************************
```

```
368                     * module: DC_TRANS
369                     ************************************************
370                     *                                              *
371                     *   procedure ENTER_DATA (                     *
372                     *               var SCT:  select_code_table;   *
373                     *               PTR:      ^ data_bytes;        *
374                     *               var COUNT:longword       );    *
375                     *                                              *
376                     *  COUNT initially passes the number of bytes  *
377                     *  which the upper level wants to read.  THE   *
378                     *  ROUTINE DOES NOT NECESSARILY READ THIS MANY! *
379                     *  Upon exit COUNT will be reflect the number  *
380                     *  of data bytes entered, whether or not there *
381                     *  is an escape.                               *
382                     *                                              *
383                     *  escape(EOD): Terminated by reaching a control*
384                     *     block when not enough bytes have been    *
385                     *     entered.  TERM&MODE may be read with     *
386                     *     STATUS 9 and 10.                         *
387                     *                                              *
388                     ************************************************
389
390
391
392
393
394
395
396                     * module: DC_TRANS
397                     ************************************************
398                     *                                              *
399                     *   procedure OUTPUT_END (                     *
400                     *               var SCT:  select_code_table );  *
401                     *                                              *
402                     *   Equivalent to the BASIC OUTPUT Sc;END.     *
403                     *                                              *
404                     *   This operation may hang waiting for space. *
405                     *                                              *
406                     ************************************************
```



```
408                     * module: DC_TRANS
409                     ************************************************
410                     *                                              *
411                     *   procedure CONTROL_BFD (                    *
412                     *               var SCT:  select_code_table;   *
413                     *               REG:      0..127;              *
414                     *               VAL:      0..255);             *
415                     *                                              *
416                     *   Control register 0 is intercepted and      *
417                     *   if MODE=0 no action is performed, otherwise *
418                     *   the card is reset IMMEDIATELY.             *
419                     *                                              *
420                     *   This operation may hang waiting for space. *
421                     *                                              *
422                     *   The ranges of REG & VAL are not checked    *
423                     *   for validity.                              *
424                     *                                              *
425                     ************************************************
426
427
428
429
430                     ************************************************
431                     *
432                     *   Unresolved problems:
433                     *
434                     *       How is the hardware ISR linked in?
435                     *
436                     *       How are registers saved/restored on int?
437                     *
438                     *       How does the ISR find the select code &
439                     *           select_code_record?
440                     *
441                     *       Should we do busybits?
442                     *
443                     *       How are overlapped transfers implemented?
444                     *
445                     ************************************************
```

```
447                     ************************************************
448                     *
449                     *   This code is required to check the validity
450                     *   of a 98628/9 card:
451                     *
452                     *   var DSDP: 0..65535;
453                     *       ATTR: 0..255;
454                     *
455                     *   if binand(readio(SC,1),60)=52 then begin
456                     *       DSDP := readio(SC,16395) * 256
457                     *                + readio(SC,16393);
458                     *       if (DSDP<32768) then begin
459                     *           ATTR := readio(SC,DSDP*2+1);
460                     *           if binand(ATTR,127)=1 then
461                     *               (*>>> CLAIM CARD! <<<*)  ;
462                     *       end;
463                     *   end;
464                     *
465                     *   Remember that the readio operations above
466                     *   can bus error!
467                     *
468                     ************************************************
469
470
471
472                     ************************************************
473                     *
474                     *   One of the arguments to all routines is a
475                     *   structure I call SCT:select_code_table.
476                     *   This is a structure allocated by the higher
477                     *   level Modcal code, the last 34 bytes of
478                     *   which I INITIALIZE AFTER MODCAL HAS
479                     *   INITIALIZED THE FRONT PART.
480                     *
481                     *   THE C_ADR FIELD MUST BE INITIALIZED BEFORE
482                     *   CALLING THIS ROUTINE:
483                     *
484                     *       procedure ALVINIT (
485                     *                   var SCT: select_code_table );
486                     *
487                     *   This routine also resets the card.  This
488                     *   routine should be used at INITIALIZE time
489                     *   but not at RESET (that's CONTROL 0;1).
490                     *
491                     ************************************************
```



```
495                     ************************************************
496                     *
497                     *   Equivalences:
498                     *
499                     *       set_serial, clear_serial
500                     *               CONTROL_BFD(REG<=8)
501                     *
502                     *       serial_line - DIRECT_STATUS(REG<=8)
503                     *
504                     *       set_baud_rate
505                     *
506                     *           ID := DIRECT_STATUS(REG<=3);
507                     *           DIRECT_CONTROL(REG<=20,VAL<=speed);
508                     *           if ID=1 then
509                     *               DIRECT_CONTROL(REG<=21,VAL<=speed);
510                     *
511                     *           ('speed' must go through a mapping!)
512                     *
513                     *       set_char_length - CONTROL_BFD(REG<=34)
514                     *
515                     *       set_stop_bits - CONTROL_BFD(REG<=35)
516                     *
517                     *       set_parity - CONTROL_BFD(REG<=36)
518                     *
519                     *       break - DIRECT_CONTROL(REG<=6)
520                     *
521                     *       abort - DIRECT_CONTROL(REG<=125,VAL<=0)
522                     *               this also aborts transfers!
523                     *
524                     *       clear - DIRECT_CONTROL(REG<=101,VAL<=0)
525                     *
526                     *       ioreset - CONTROL_BFD(REG<=0,VAL<=1)
527                     *
528                     *       readchar - ENTER_DATA(COUNT<=1)
529                     *               (ENTER_DATA will terminate on
530                     *               a control block!!!)
531                     *
532                     *       writechar - OUTPUT_DATA(COUNT<=1)
533                     *
534                     *       set_timeout - write to value in
535                     *               select_code_table (units=1ms)
536                     *
537                     *       LEVEL 2 functions - supersets of readchar
538                     *               & writechar
539                     *
540                     *       handshake transfer - ENTER_DATA or
541                     *               OUTPUT_DATA (ENTER_DATA will
542                     *               terminate on a control block!!!)
543                     *
544                     *       outbound transfer_end - OUTPUT_DATA then
545                     *               OUTPUT_END
546                     *
547                     *       inbound transfer_end - ENTER_DATA
548                     *
549                     *       overlapped transfers - set up the transfer
                                control block and then call
                                START_TRANSFER_IN or
```

```
550                         *               START_TRANSFER_OUT.
551                         *
552                         ************************************************
553
554                             list
```



```
557                             include COMDCL
```

```
560                     ********************************************************************************
561                     *
562                     *       modified:  02/22/82  JPC   added parm to user EOT & ISR proc's
563                     *                  08/01/83  JS    added timer_present and sysflag2 equ's
564                     *
565                     ********************************************************************************
566                     *
567                     *       HPL CONVENTIONS
568                     *
569                     *
570                     *         Much of this code is taken intact from the 9826 HPL
571                     *         language system EIO ROM ( extended I/O ROM ).
572                     *
573                     *         The Pascal that will be calling this code uses
574                     *         the stack for parameter passage.  The HPL code
575                     *         uses the Ax and Dx registers for all parameters.
576                     *         The Pascal driver entry points on the previous pages
577                     *         take care of getting the parameters into the correct
578                     *         registers.
579                     *
580                     *
581                     *       GENERAL HPL ENTRY/EXIT CONDITIONS:
582                     *
583                     *         A1.L = CARD ADDRESS
584                     *         A2.L = DRIVER TEMP ADDRESS
585                     *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
586                     *
587                     *
588                     *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
589                     *
590                     *         A3.L = BUFFER CONTROL BLOCK ADDRESS
591                     *                In addition to the A1/A2 convention, Pascal will use
592                     *                A3 for a pointer to the buffer control block.
593                     *                The HPL system kept much of the transfer
594                     *                information in the s.c. temps.
595                     *
596                     *         TIMEOUT(A2) = contains timeout information
597                     *                Timeout was a global temp in HPL and a timeout
598                     *                generated an error.
599                     *                In PASCAL each card has a timeout value stored in
600                     *                its temporary area.  A timeout error
601                     *                generates an ESCAPE ( which can be trapped ).
602                     *
603                     *
604                     ********************************************************************************
```



```
606                     ********************************************************************************
607                     *
608                     *
609                     *         DRIVER TEMPS TEMPLATE
610                     *
611                     *           OFFSET FROM A2
612                     *
613                     *           HPL DECLARATIONS ( MODIFIED )
614                     *
615                     *
616                     ********************************************************************************
617           0000 0000 ISR_ENTRY     EQU 0       ..19        PASCAL ISR LINK & UNLINK area
618           0000 0014 USER_ISR      EQU 20                  user ISR:  do NOT change the proc/stat link/parm ordering!!!
619           0000 0014 H_ISR_PR      EQU 20      ..23            procedure ptr
620           0000 0018 H_ISR_SL      EQU 24      ..27            static link
621           0000 001C H_ISR_PM      EQU 28      ..31            parameter
622           0000 0020 C_ADR         EQU 32      ..35        card address
623           0000 0024 BUFI_OFF      EQU 36      ..39        buffer pointer offset
624           0000 0028 BUFO_OFF      EQU 40      ..43        buffer pointer offset
625           0000 002C EIRB_OFF      EQU 44                  eir byte
626           0000 002D IO_SC         EQU 45                  select code ( i.e. 7, 22, etc. )
627           0000 002E TIMEOUT       EQU 46      ..49        timeout value
628                     *                                       =0 : no timeout
629                     *                                      #0 : value of timeout
630           0000 0032 MA_W          EQU 50      ..51        word access to my address
631           0000 0033 MA            EQU 51                  byte access to my address
632           0000 0034 AVAIL_OFF     EQU 52      ..??        standard space taken from temps
633                     *                 52      ..83        normal cards ( 32  bytes )
634                     *                 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 FFF8 RCVR_BLK      EQU    SYSGLOBALS-10

0000 0001 TIMER_PRESENT EQU    1        JS  8/1/83  SYSFLAG2 BIT -- 0=>TIMER PRESENT
FFFF FEDA SYSFLAG2      EQU    $FFFFFEDA          JS  8/1/83

                        list
```

```
738
739
740                     ********************************************************************************
741                     *                          Data Comm card RAM locations                        *
742                     *                       (byte offsets from base address)                       *
743
744             0000 0000 RESET_ID          equ     $000000
745             0000 0002 INT_DMA           equ     $000002
746             0000 0004 SEMAPHORE         equ     $000004
747             0000 4000 INT_COND          equ     $004000
748             0000 4002 COMMAND           equ     $004002
749             0000 4004 DATA_REG          equ     $004004
750             0000 4006 PRIMARY_ADDR      equ     $004006
751             0000 4008 DSDP              equ     $004008
752             0000 400C ERROR_CODE        equ     $00400C
753
754                     ************************** Data Structures Descriptor ***************************
755
756             0000 0000 ATTRIBUTES        equ     $000000
757             0000 0002 TR_QUEUE_ADDR     equ     $000002
758             0000 0006 PRIM_0_ADDR       equ     $000006
759
760                     ********************************** Queue ************************************************
761
762             0000 0000 TXENDBLOCKSPACE equ       $000000
763             0000 0001 RXDATABUFF_NUMB equ       $000001
764             0000 0004 TXBUFF            equ     $000004
765             0000 0024 RXBUFF            equ     $000024
766
767             0000 0000 CTRL_AREA         equ     $000000
768             0000 0010 DATA_AREA         equ     $000010
769
770                     ******************************** Buffer record ***********************************
771
772             0000 0000 ADDR              equ     $000000
773             0000 0004 SIZE              equ     $000004
774             0000 0008 FILL              equ     $000008
775             0000 000C EMPTY             equ     $00000C
776
777                     ******************************** Control block ***********************************
778
779             0000 0000 POINTER           equ     $000000
780             0000 0004 TERMFIELD         equ     $000004
781             0000 0006 MODEFIELD         equ     $000006
782             0000 0008 CTRLBLKSIZE       equ     $000008
783
784                     ******************************** select_code_table ********************************
785
786             0000 0034 ovrlaper          equ AVAIL_OFF+00 .. 1    ; word
787             0000 0036 usr0mask          equ AVAIL_OFF+02         ; byte
788             0000 0037 which_RXbuf       equ AVAIL_OFF+03         ; byte
789             0000 0038 last_enter_term equ AVAIL_OFF+04           ; byte
790             0000 0039 last_enter_mode equ AVAIL_OFF+05           ; byte
791             0000 003A intbits           equ AVAIL_OFF+06         ; 8 bits
792                     * unused                    07
793                     * The following 26 bytes are saved at interrupt
794             0000 003C term_and_mode     equ AVAIL_OFF+08 .. 09  ; Encompasses the two below
```



```
795             0000 003C term              equ AVAIL_OFF+08         ; byte
796             0000 003D mode              equ AVAIL_OFF+09         ; byte
797             0000 003E data_address      equ AVAIL_OFF+10 .. 13   ; pointer
798             0000 0042 data_number       equ AVAIL_OFF+14 .. 17   ; integer
799             0000 0046 outer_tx_count    equ AVAIL_OFF+18 .. 21   ; integer
800             0000 004A timeout_counter equ AVAIL_OFF+22 .. 25     ; integer
801             0000 004E inner_counter     equ AVAIL_OFF+26 .. 29   ; integer
802             0000 0052 inner_tx_count    equ AVAIL_OFF+30 .. 33   ; integer
803                     * This is where they are saved:
804             0000 0056 int_savespace     equ AVAIL_OFF+34 .. 59
805             0000 0070 SR_image          equ AVAIL_OFF+60 .. 61   ; word
806             0000 0072 RCR_hook          equ AVAIL_OFF+62 .. 69   ; procedure
807             0000 007A err_hook          equ AVAIL_OFF+70 .. 77   ; procedure
808             0000 0082 trc_hook          equ AVAIL_OFF+78 .. 85   ; procedure
809             0000 008A bit6_hook         equ AVAIL_OFF+86 .. 93   ; procedure
810             0000 0092 bit7_hook         equ AVAIL_OFF+94 .. 101  ; procedure
811
812
813
814             0000 009A sctablebytes      equ AVAIL_OFF+102        ; size of the entire table for allocation
815
816             0000 0000 error_int         equ 0   ; Bits for interrupt
817             0000 0001 rx_int            equ 1   ; register
818             0000 0002 tx_int            equ 2
819             0000 0003 ON_INTR_int       equ 3
820             0000 0004 RC_reset_int      equ 4
821             0000 0005 trace_int         equ 5
822
823                     ********************************************************************************
824
825                         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:                                                              *
                        *              a3.1 =  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 RELEASE_ACCESS MUST BE      *
                        *               CALLED AT THE SAME LEVEL ON THE STACK!!!!!                     *
                        *                                                                              *
                        *       This bashes d2.1.                                                      *
                        *                                                                              *
                        ********************************************************************************

                  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 0838 0001        btst    #timer_present,sysflag2 ' check if timer present    tttt JS
         FEDA
0000003A 6718             beq.s   gatimed                   if so then go use it      tttt JS
0000003C 243C 0002        move.l  #157500,d2 [CALIBRATED 1 SEC] ; Initialize counter
         673C
00000042 4A2B 0004 galoop tst.b   SEMAPHORE(a3)           ; Fetch semaphore bit in bit 7 (sign)
00000046 6A5C             bpl.s   gadone                  ; If bit 7 true then done!
00000048 5382             subq.l  #1,d2                   ; Loop for preset time
0000004A 66F6             bne.s   galoop

                        *                                 Timed out: escape, but first...
0000004C 584F      gaterr addq    #4,sp                   pop return addres               scs
```



```
0000004E 46DF             move    (sp)+,sr                restore user mode               scs
                  * scs   move    4(sp),sr                ; Get old SR back
00000050 6000 0306        bra     lunched                 ; Now escape
                  *
00000054 4A2B 0004 gatimed tst.b  semaphore(a3)           first do quick test         tttt JS
00000058 6A4A             bpl.s   gadone                  before timing               tttt JS
0000005A 1F3C 0001        move.b  #1,-(sp)                set up timing record        tttt JS
0000005E 2F3C 0000        move.l  #1000,-(sp)             MS followed by boolean      tttt JS
         03E8
00000034 243C 0000 gatlp1 move.l  #254,d2                 quick loop -- 1 ms @ 16 MHz tttt JS
         00FE
0000003A 4A2B 0004 gatlp2 tst.b   semaphore(a3)           check semaphore             tttt JS
0000003E 6A12             bpl.s   gatexit                 if ok then get out          tttt JS
00000070 5382             subq.l  #1,d2                   else hang in tight loop     tttt JS
00000072 66F6             bne     gatlp2                                              tttt JS
00000074 4857             pea     (sp)                    now check the timer         tttt JS
00000076 4EB9 0000        jsr     check_timer             parameter is ptr to record  tttt JS
         0000
0000007C 6AE6             bpl     gatlp1                  if not timeout do tight loop tttt JS
0000007E 5C4F             addq    #6,sp                   else fix the stack          tttt JS
00000080 60CA             bra     gaterr                  and do timeout escape       tttt JS
00000082 5C4F      gatexit addq   #6,sp                   normal exit is here         tttt JS
00000084 4E75             rts                                                         tttt JS
```

```
                        ********************************************************************************
                        *                                                                              *
                        *       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 0086 release_access equ *
00000086 1740 0004              move.b  d0,SEMAPHORE(a3)        ; Store don't-care into semaphore.

0000008A 3F2F 0004              move.w  4(sp),-(sp)             switch SR and return address    scs
0000008E 4A38 0000              tst.b   m68ktype                is this a 68000?          tttt JS
00000092 660A                   bne.s   release_10or12          br if not                 tttt JS
00000094 2F6F 0002              move.l  2(sp),4(sp)                                               scs
              0004
0000009A 3E9F                   move.w  (sp)+,(sp)                                                scs
0000009C 4E73                   rte                             restore user mode, return         scs
                        *
              0000 009E release_10or12 equ *
0000009E 426F 0006              clr.w   6(sp)           fake vector offset word where SR was tttt JS
000000A2 4E73                   rte                       and do an rte back to user mode     tttt JS

000000A4 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 00A6 find_TRBUF equ *
000000A6 7A00                   moveq   #0,d5
000000A8 247C 0000              movea.l #TR_QUEUE_ADDR,a2
              0002
000000AE 600C                   bra.s   findTR

              0000 00B0 find_TXBUF equ *
000000B0 7A04                   moveq   #TXBUFF,d5
000000B2 6002                   bra.s   find

              0000 00B4 find_RXBUF equ *
000000B4 7A24                   moveq   #RXBUFF,d5
000000B6 247C 0000 find         movea.l #PRIM_0_ADDR,a2
              0006

000000BC 4284           findTR  clr.l   d4
000000BE 0A2B 4008              movep.w DSDP(a3),d4
000000C2 EE5C                   ror.w   #7,d4
000000C4 D88B                   add.l   a3,d4
000000C6 2244                   movea.l d4,a1                   ; a1 points to Data Struct Descriptor

000000C8 4284                   clr.l   d4
000000CA D3CA                   adda.l  a2,a1                   ; add offset to which queue table
000000CC 0929 0000              movep.w 0(a1),d4
000000D0 EE5C                   ror.w   #7,d4
000000D2 D88B                   add.l   a3,d4
000000D4 D885                   add.l   d5,d4
000000D6 2444                   movea.l d4,a2                   ; a2 points to buffer record
000000D8 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 00DA find_RX_DATA equ *
000000DA 227C 0000          movea.l #DATA_AREA,a1
            0010
000000E0 4285               clr.l   d5                      ; compute offset for WHICH rx data
000000E2 1A2C 0037          move.b  which_RXbuf(a4),d5      ; buffer we are using
000000E6 E985               asl.l   #4,d5
000000E8 D3C5               adda.l  d5,a1

000000EA 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 aZ) 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 aZ) 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 00EC find_DATA_AREA equ *
000000EC 227C 0000          movea.l #DATA_AREA,a1
            0010
000000F2 6006               bra.s   findare

            0000 00F4 find_CTRL_AREA equ *
000000F4 227C 0000          movea.l #CTRL_AREA,a1
            0000

000000FA D3CA       findare adda.l  a2,a1                   ; a1 points to data/ctrl part of record
000000FC 4285               clr.l   d5
000000FE 0B09 0004          movep.w SIZE(a1),d5
00000102 E05D               ror.w   #8,d5                   ; d5 = SIZE in bytes

00000104 4284               clr.l   d4
00000106 0909 0000          movep.w ADDR(a1),d4
0000010A EE5C               ror.w   #7,d4
0000010C D88B               add.l   a3,d4
0000010E 2244               movea.l d4,a1                   ; a1 points to front of buffer area
00000110 D885               add.l   d5,d4
00000112 D885               add.l   d5,d4                   ; d4 points past end of buffer area

00000114 4E75               rts
```

```
1068
1069
1070                    *       ***     ***    *   *   *   *    ***    *   *   ****
1071                    *      *   *   *   *   ** **   ** **   *   *   **  *   *   *
1072                    *          *   *   *   * * *   * * *   *   *   * * *   *   *
1073                    *          *   *   *   *   *   *   *   *****   *  **   *   *
1074                    *          *   *   *   *   *   *   *   *   *   *   *   *   *
1075                    *      *   *   *   *   *   *   *   *   *   *   *   *   *   *
1076                    *       ***     ***    *   *   *   *   *   *   *   *   ****
1077
1078
1079
1080
1081
1082                    ********************************************************************************
1083                    *                                                                              *
1084                    *   procedure ALVINIT(var SCT: select_code_table)                              *
1085                    *                                                                              *
1086                    ********************************************************************************
1087
1088                 0000 0116 alvinit equ     *
1089    00000116 205F               movea.l (sp)+,a0
1090    00000118 285F               movea.l (sp)+,a4
1091    0000011A 4850               pea     (a0)
1092    0000011C 266C 0020          movea.l c_adr(a4),a3
1093    00000120 528B               addq.l  #1,a3
1094
1095    00000122 243C 0000          move.l  #sctablebytes-1-AVAIL_OFF,d2
                     0065
1096    00000128 4234 2034 clrloop clr.b   AVAIL_OFF(a4,d2)
1097    0000012C 51CA FFFA          dbf     d2,clrloop
1098
1099    00000130 102B 0002          move.b  INT_DMA(a3),d0  ; This SR is used to
1100    00000134 0240 0030          andi.w  #$0030,d0       ; set the interrupt level
1101    00000138 0640 0230          addi.w  #$0230,d0       ; equal to that of the
1102    0000013C E940               asl.w   #4,d0           ; card.
1103    0000013E 3940 0070          move.w  d0,SR_image(a4)
1104
1105    00000142 422C 0037          CLR.B   WHICH_RXBUF(A4) ;                              ( SPRyyy TM 6/15/82 )
1106
1107    00000146 6100 0286          bsr     chk_err         ; See if card is giving
1108                             *                          ; overlapped error
1109    0000014A 6100 01D0          bsr     do_reset        ; This can escape if card bad
1110
1111    0000014E 6000 01EC          bra     check_ov_error  ; Escape if any error
```



```
1113                    ********************************************************************************
1114                    *                                                                              *
1115                    *       routine eir:    Enables interrupts on this card only.                  *
1116                    *               ===                                                            *
1117                    *                                                                              *
1118                    *       At entry:                                                              *
1119                    *               a3.l =  card's base address ($00xx0001)                        *
1120                    *                                                                              *
1121                    *       Upon normal exit:                                                      *
1122                    *               No registers are changed.                                      *
1123                    *                                                                              *
1124                    ********************************************************************************
1125
1126                 0000 0152 eir     equ     *
1127    00000152 117C 0080          move.b  #$80,INT_DMA(a3)                ; Set enable-interrupt bit true
                     0002
1128    00000158 4E75               rts
1129
1130
1131                    ********************************************************************************
1132                    *                                                                              *
1133                    *       routine dir:    Disables interrupts on this card only.                 *
1134                    *               ===                                                            *
1135                    *                                                                              *
1136                    *       At entry:                                                              *
1137                    *               a3.l =  card's base address ($00xx0001)                        *
1138                    *                                                                              *
1139                    *       Upon normal exit:                                                      *
1140                    *               No registers are changed.                                      *
1141                    *                                                                              *
1142                    ********************************************************************************
1143
1144                 0000 015A dir     equ     *
1145    0000015A 422B 0002          clr.b   INT_DMA(a3)                     ; Clear enable-interrupt bit
1146    0000015E 4E75               rts
```

```
1148                        ************************************************
1149                        *                                              *
1150                        *   procedure DIRECT_CONTROL (                 *
1151                        *               var SCT:  select_code_table;   *
1152                        *               REG:      1..127 & 256 & 257;  *
1153                        *               VAL:      0..255  );           *
1154                        *                                              *
1155                        ************************************************
1156
1157             0000 0160 direct_control equ *
1158    00000160 205F               movea.l (sp)+,a0
1159    00000162 321F               move.w  (sp)+,d1        ; VAL
1160    00000164 341F               move.w  (sp)+,d2        ; REG
1161    00000166 285F               movea.l (sp)+,a4        ; SCT
1162    00000168 4850               pea     (a0)
1163    0000016A 266C 0020          movea.l c_adr(a4),a3
1164    0000016E 528B               addq.l  #1,a3
1165
1166    00000170 6100 01CA          bsr     check_ov_error  ; Escape if any error
1167
1168    00000174 1741 4004          move.b  d1,DATA_REG(a3) ; Send VAL
1169    00000178 1742 4002          move.b  d2,COMMAND(a3)  ; then REG
1170    0000017C B43C 007D          cmp.b   #125,d2
1171    00000180 6608               bne.s   not125
1172    00000182 6100 047C          bsr     outxfr_done     ; For ctrl 125 (abort)
1173    00000186 6100 048A          bsr     inxfr_done      ; abort transfers
1174             0000 018A not125   equ   *
1175                        *                                   (tm) mod - 12/02/81
1176    0000018A B47C 0100          CMP.W   #256,D2             (tm) CHECK FOR ABORT TFR IN
1177    0000018E 6700 0482          BEQ     INXFR_DONE          (tm)
1178    00000192 B47C 0101          CMP.W   #257,D2             (tm) CHECK FOR ABORT TFR OUT
1179    00000196 6700 0468          BEQ     OUTXFR_DONE         (tm)
1180
1181    0000019A 0838 0001          btst    #timer_present,sysflag2   check if timer present  tttt JS
                FEDA
1182    000001A0 6718               beq.s   ctltime                   if so then go use it    tttt JS
1183
1184    000001A2 203C 0002          move.l  #181851,d0 [CALIBRATED 1 SEC]   ; Now start counter for timeout
                C65B
1185    000001A8 4A2B 4002 ctloop   tst.b   COMMAND(a3)
1186    000001AC 6708               beq.s   ctldun          ; Done when COMMAND=0
1187    000001AE 5380               subq.l  #1,d0
1188    000001B0 66F6               bne.s   ctloop                      ; Otherwise decrement counter
1189    000001B2 6000 01A4          bra     lunched                     ; & escape
1190
1191    000001B6 6000 0184 ctldun   bra     check_ov_error
1192
1193    000001BA 1F3C 0001 ctltime move.b   #1,-(sp)        set up timer record           tttt JS 8/11/83
1194    000001BE 2F3C 0000          move.l  #1000,-(sp)                               tttt JS 8/11/83
                03E8
1195    000001C4 4A2B 4002 ctltlop tst.b    command(a3)     see if done                   tttt JS 8/11/83
1196    000001C8 6710               beq.s   ctlexit         if so then return             tttt JS 8/11/83
1197    000001CA 4857               pea     (sp)            else push pointer to time rec tttt JS 8/11/83
1198    000001CC 4EB9 0000          jsr     check_timer     and see if timed out yet      tttt JS 8/11/83
                0000
1199    000001D2 6AF0               bpl     ctltlop         no -- keep trying             tttt JS 8/11/83
1200    000001D4 5C4F               addq    #6,sp           yes, give one more try        tttt JS 5/3/84
```



```
1201    000001D6 700A               moveq   #10,d0          using short count             tttt JS 5/3/84
1202    000001D8 60CE               bra     ctloop          in normal timing loop         tttt JS 5/3/84
1203    000001DA 5C4F       ctlexit addq    #6,sp           normal exit -- clean stack    tttt JS 8/11/83
1204    000001DC 6000 015E          bra     check_ov_error  and go check errors           tttt JS 8/11/83
```

```
1206                        ************************************************
1207                        *                                              *
1208                        *   procedure DIRECT_STATUS (                  *
1209                        *               var SCT:  select_code_table;   *
1210                        *               REG:      1..127;              *
1211                        *               var VAL:  word    );           *
1212                        *                                              *
1213                        *   These registers are intercepted:           *
1214                        *       0:  Gives value from RESET_ID          *
1215                        *       1:  Returns true if hardware interrupts *
1216                        *       2:  Returns bit 2 = in xfr active;     *
1217                        *                   bit 3 = out xfr active     *
1218                        *       5:  Returns 2 bits saying state of Rx  *
1219                        *           buffer                             *
1220                        *       9:  Returns last ENTER TERM            *
1221                        *       10: Returns last ENTER MODE            *
1222                        *       11: Returns # bytes available in Tx    *
1223                        *           queue, or 0 if there's not 3       *
1224                        *           control block positions available  *
1225                        *                                              *
1226                        ************************************************
1227
1228             0000 01E0 direct_status equ *
1229    000001E0 205F                movea.l (sp)+,a0
1230    000001E2 225F                movea.l (sp)+,a1        ; addr(VAL)
1231    000001E4 301F                move.w  (sp)+,d0        ; REG
1232    000001E6 285F                movea.l (sp)+,a4        ; SCT
1233    000001E8 4850                pea     (a0)
1234    000001EA 266C 0020           movea.l c_adr(a4),a3
1235    000001EE 528B                addq.l  #1,a3
1236
1237    000001F0 6100 014A           bsr     check_ov_error  ; Escape if any error
1238
1239    000001F4 4241                clr.w   d1              ; d1 will hold result
1240    000001F6 2F09                move.l  a1,-(sp)
1241
1242    000001F8 4A00                tst.b   d0
1243    000001FA 6738                beq.s   sts0
1244    000001FC 0C00 0002           cmpi.b  #2,d0           ; Check for intercepted regs
1245    00000200 6B38                blt.s   sts1
1246    00000202 6744                beq.s   sts2
1247    00000204 0C00 0005           cmpi.b  #5,d0
1248    00000208 6758                beq.s   sts5
1249    0000020A 0C00 0009           cmpi.b  #9,d0
1250    0000020E 6766                beq.s   sts9
1251    00000210 0C00 000A           cmpi.b  #10,d0
1252    00000214 6766                beq.s   sts10
1253    00000216 0C00 000B           cmpi.b  #11,d0
1254    0000021A 6766                beq.s   sts11
1255    0000021C 0C3C 0080           add.b   #128,d0
1256    00000220 1940 003C           move.b  d0,term(a4)     ; Send TERM
1257
1258    00000224 6100 0092           bsr     direct_command
1259    00000228 122C 003D           move.b  mode(a4),d1
1260
1261    0000022C 225F        gotsts  movea.l (sp)+,a1
1262    0000022E 3381                move.w  d1,(a1)                 ; Return value
```



```
1263    00000230 6000 010A           bra     check_ov_error
1264
1265                        *
1266                        *  Special intercepted registers
1267                        *
1268
1269    00000234 122B 0000 sts0      move.b  RESET_ID(a3),d1
1270    00000238 60F2                bra.s   gotsts
1271
1272    0000023A 082B 0007 sts1      btst    #7,INT_DMA(a3)
            0002
1273    00000240 56C1                sne     d1
1274    00000242 C27C 0001           and.w   #$0001,d1
1275    00000246 60E4                bra.s   gotsts
1276
1277    00000248 246C 0024 sts2      movea.l BUFI_OFF(a4),a2
1278    0000024C 200A                move.l  a2,d0
1279    0000024E 6704                beq.s   sts2a
1280    00000250 08C1 0002           bset    #2,d1
1281    00000254 246C 0028 sts2a     movea.l BUFO_OFF(a4),a2
1282    00000258 200A                move.l  a2,d0
1283    0000025A 67D0                beq.s   gotsts
1284    0000025C 08C1 0003           bset    #3,d1
1285    00000260 60CA                bra.s   gotsts
1286
1287    00000262 6100 FE50 sts5      bsr     find_RXBUF
1288    00000266 6100 FEF2           bsr     dir
1289    0000026A 6100 0580           bsr     RX_stuff_avail
1290    0000026E 3200                move.w  d0,d1
1291    00000270 6100 FEE0           bsr     eir
1292    00000274 60B6                bra.s   gotsts
1293
1294    00000276 122C 0038 sts9      move.b  last_enter_term(a4),d1
1295    0000027A 60B0                bra.s   gotsts
1296
1297    0000027C 122C 0039 sts10     move.b  last_enter_mode(a4),d1
1298    00000280 60AA                bra.s   gotsts
1299
1300    00000282 6100 FE2C sts11     bsr     find_TXBUF
1301    00000286 6100 FED2           bsr     dir
1302    0000028A 6100 FE68           bsr     find_CTRL_AREA
1303    0000028E 6100 0840           bsr     TXCTRLBUFFroom
1304    00000292 4241                clr.w   d1
1305    00000294 0483 0000           subi.l  #12,d3
            000C
1306    0000029A 6090                blt.s   gotsts
1307    0000029C 6100 FE4E           bsr     find_DATA_AREA
1308    000002A0 6100 0840           bsr     TXDATABUFFroom
1309    000002A4 3203                move.w  d3,d1
1310    000002A6 6100 FEAA           bsr     eir
1311    000002AA 6080                bra     gotsts
```

```
1313                    ********************************************************************************
1314                    *                                                                              *
1315                    *       routine put_INTMASK:    Sends the value in usr0mask to the card.       *
1316                    *                  ==========                                                  *
1317                    *                                                                              *
1318                    *       At entry:                                                              *
1319                    *               a3.l =  card base address ($00xx000i)                          *
1320                    *               a4.l =  address of sc_subtabletype structure                   *
1321                    *               sc_subtabletype.usr0mask has value to send to card.            *
1322                    *                                                                              *
1323                    *       This bashes term and mode in the select code subtable.                 *
1324                    *                                                                              *
1325                    *   This bashes d0.                                                            *
1326                    *                                                                              *
1327                    ********************************************************************************
1328
1329            0000 02AC put_INTMASK equ *
1330    000002AC 197C 0079      move.b  #121,term(a4)           ; Send the new driver interrupt mask
                 003C
1331    000002B2 196C 0036      move.b  usr0mask(a4),mode(a4)   ; down with control #121
                 003D
1332                    ***     bra     direct_command
1333
1334                    ********************************************************************************
1335                    *                                                                              *
1336                    *   routine direct_command                                                     *
1337                    *           ==============                                                     *
1338                    *                                                                              *
1339                    *       Uses:   a3.l =  Base address of card                                   *
1340                    *               a4.l =  Address of SCT: select_code_table                      *
1341                    *                                                                              *
1342                    *   This bashes d0.                                                            *
1343                    *                                                                              *
1344                    ********************************************************************************
1345
1346            0000 02B8 direct_command equ *
1347    000002B8 176C 003D      move.b  mode(a4),DATA_REG(a3)
                 4004
1348    000002BE 176C 003C      move.b  term(a4),COMMAND(a3)    ; Send TERM
                 4002
1349
1350    000002C4 0838 0001      btst    #timer_present,sysflag2   is timer available?  tttt JS 8/11/83
                 FEDA
1351    000002CA 671C           beq.s   dctime                    if so go use it      tttt JS 8/11/83
1352
1353    000002CC 203C 0002      move.l  #181851,d0 [CALIBRATED 1 SEC]   ; Now start counter for timeout
                 C65B
1354    000002D2 4A2B 4002 dcloop  tst.b   COMMAND(a3)
1355    000002D6 6708           beq.s   dcdone                    ; Done when COMMAND=0
1356    000002D8 5380           subq.l  #1,d0
1357    000002DA 66F6           bne.s   dcloop                    ; Otherwise decrement counter
1358    000002DC 6000 007A      bra     lunched                    ; & escape
1359
1360    000002E0 196B 4004 dcdone  move.b  DATA_REG(a3),mode(a4)
                 003D
1361    000002E6 4E75           rts
1362
```



```
1363    000002E8 1F3C 0001 dctime  move.b  #1,-(sp)              set up timer record  tttt JS 8/11/83
1364    000002EC 2F3C 0000      move.l  #1000,-(sp)             for 1 sec wait       tttt JS 8/11/83
                 03E8
1365    000002F2 4A2B 4002 dctloop tst.b   command(a3)          see if done          tttt JS 8/11/83
1366    000002F6 6710           beq.s   dctexit                 if so then return    tttt JS 8/11/83
1367    000002F8 4857           pea     (sp)                    check timer          tttt JS 8/11/83
1368    000002FA 4EB9 0000      jsr     check_timer                                  tttt JS 8/11/83
                 0000
1369    00000300 6AF0           bpl     dctloop                 if not timeout, br   tttt JS 8/11/83
1370    00000302 5C4F           addq    #6,sp                   timeout, clean stk   tttt JS 5/3/84
1371    00000304 700A           moveq   #10,d0                  and try once more    tttt JS 5/3/84
1372    00000306 60CA           bra     dcloop                  with short count     tttt JS 5/3/84
1373    00000308 5C4F       dctexit addq    #6,sp               normal exit, cleanup tttt JS 8/11/83
1374    0000030A 60D4           bra     dcdone                  and return           tttt JS 8/11/83
1375
```

```
1377                    ********************************************************************************
1378                    *                                                                              *
1379                    *       routine get_INTMASK:    Reads the value of usr0mask from the card.     *
1380                    *                 ===========                                                  *
1381                    *                                                                              *
1382                    *       At entry:                                                              *
1383                    *               a3.l =  card base address ($00xx0001)                          *
1384                    *               a4.l =  address of sc_subtabletype structure                   *
1385                    *                                                                              *
1386                    *       At exit:                                                               *
1387                    *               sc_subtabletype.usr0mask has value from the card.              *
1388                    *                                                                              *
1389                    *       This pashes term and mode in the select code subtable.                 *
1390                    *                                                                              *
1391                    *   This bashes d0.                                                            *
1392                    *                                                                              *
1393                    ********************************************************************************
1394
1395          0000 030C get_INTMASK equ *
1396    0000030C 117C 00F9      move.b  #121+128,term(a4)       ; Get the current interrupt mask
          030C
1397    00000312 61A4           bsr     direct_command
1398    00000314 116C 003D      move.b  mode(a4),usr0mask(a4)   ; from register #121
          0036
1399    0000031A 4E75           rts
1400
1401
1402                    ********************************************************************************
1403                    *                                                                              *
1404                    *       do_reset:  resets the card, waits for it to complete powerup and       *
1405                    *       ========   then gets INTMASK again.                                    *
1406                    *                                                                              *
1407                    *       Uses:   a3.l =  Base address of card                                   *
1408                    *               a4.l =  Address of SCT: select_code_table                      *
1409                    *                                                                              *
1410                    *   This leaves interrupts ENABLED                                             *
1411                    *                                                                              *
1412                    *   This bashes d0.                                                            *
1413                    *                                                                              *
1414                    ********************************************************************************
1415
1416
1417          0000 031C do_reset equ    *
1418    0000031C 6100 02E2      bsr     outxfr_done     ; Abort transfers
1419    00000320 6100 02F0      bsr     inxfr_done
1420    00000324 6100 FE34      bsr     dir                     ; Disable card interrupts to prevent
1421                    *                                       ; conflicts
1422    00000328 117C 0080      move.b  #$80,RESET_ID(a3)       ; Send reset ($80) to card
          0000
1423    0000032E 6100 FCFA      bsr     gain_access             ; Wait until SEMAPHORE is freed
1424    00000332 6100 FD52      bsr     release_access          ; and then give it back
1425    00000336 6104           bsr     get_INTMASK
1426    00000338 6000 FE18      bra     eir
```



```
1428                    ********************************************************************************
1429                    *                                                                              *
1430                    *   routine check_ov_error:  If the 'ovrlaper' location is nonzero then        *
1431                    *            ==============  this escapes with that error.                     *
1432                    *                                                                              *
1433                    *       Uses:   a3.l =  Base address of card ($00xx0001)                       *
1434                    *               a4.l =  Address of SCT: select_code_table                      *
1435                    *                                                                              *
1436                    *   This leaves interrupts ENABLED                                             *
1437                    *                                                                              *
1438                    ********************************************************************************
1439
1440
1441          0000 033C check_ov_error equ *
1442    0000033C 6100 FE1C      bsr     dir
1443    00000340 2F00           move.l  d0,-(sp)
1444    00000342 4280           clr.l   d0
1445    00000344 302C 0034      move.w  ovrlaper(a4),d0
1446    00000348 42AC 0034      clr.w   ovrlaper(a4)
1447    0000034C 6100 FE04      bsr     eir
1448    00000350 4A40           tst.w   d0
1449    00000352 660A           bne.s   escape
1450    00000354 201F           move.l  (sp)+,d0
1451    00000356 4E75           rts
1452
1453
1454                    ****************************** Escapes ****************************************
1455
1456          0000 0358 lunched equ     *
1457    00000358 7015           moveq   #crd_dwn,d0
1458    0000035A 6002           bra.s   escape
1459
1460          0000 035C time_err equ    *
1461    0000035C 7011           moveq   #tmo_err,d0
1462
1463                    ********************************************************************************
1464                    *                                                                              *
1465                    *       routine escape: performs Pascal "escape" function.  Error exit         *
1466                    *                ======  number is to be passed in d0.                         *
1467                    *                                                                              *
1468                    *       Uses:   a3.l =  Base address of card `                                 *
1469                    *               a4.l =  Address of SCT: select_code_table                      *
1470                    *               a5.l =  Global pointer for escape arguments                    *
1471                    *               d0.l =  Escape number                                          *
1472                    *                                                                              *
1473                    *   This leaves interrupts ENABLED                                             *
1474                    *                                                                              *
1475                    ********************************************************************************
1476
1477          0000 035E escape  equ     *               ; Escape point for errors
1478    0000035E 2B40 FFBE      move.l  d0,IOE_RSLT(a5)
1479    00000362 4285           clr.l   d5              ; Tim's magic escape stuff
1480    00000364 1A2C 002D      move.b  IO_SC(a4),d5
1481    00000368 2B45 FFBA      move.l  d5,IOE_SC(a5)
1482    0000036C 3B7C FFE6      move.w  #IOE_ERROR,ESC_CODE(a5)
          FFFE
```

```
1483    00000372 177C 0080          move.b  #$80,INT_DMA(a3) ;Re-enable card interrupts if off
                 0002
1484    00000378 4E4A               trap    #10
1485
```



```
1488
1489
1490                        *       *****   *   *   *****   *****   ****    ****    *   *   ****    *****
1491                        *         *     **  *     *     *       *   *   *   *   *   *   *   *     *
1492                        *         *     * * *     *     *       *   *   *   *   *   *   *   *     *
1493                        *         *     *  **     *     ***     ****    ****    *   *   ****      *
1494                        *         *     *   *     *     *       *  *    *  *    *   *   *         *
1495                        *         *     *   *     *     *       *   *   *   *   *   *   *         *
1496                        *       *****   *   *     *     *****   *   *   *   *    ***    *         *
1497
1498
1499
1500
1501
1502                        ********************************************************************************
1503                        *                                                                              *
1504                        *                  DATA COMM CARD TOP LEVEL INTERRUPT SERVICE ROUTINE           *
1505                        *                                                                              *
1506                        *       This is reached thru the softpoll table for the appropriate            *
1507                        *       interrupt level.                                                       *
1508                        *                                                                              *
1509                        *       This handles a hardware interrupt from the data comm card.  First it   *
1510                        *       enquires to find out what interrupt conditions are pending, then it    *
1511                        *       calls the appropriate routines to handle the conditions.               *
1512                        *                                                                              *
1513                        *       At entry:                                                              *
1514                        *               a5.l =  pointer to globals area                                *
1515                        *                                                                              *
1516                        *       During this routine:                                                   *
1517                        *               a3.l =  card's base address ($00xx0001)                        *
1518                        *               a4.l =  address of sc_subtabletype structure                   *
1519                        *                                                                              *
1520                        ********************************************************************************
1521
1522             0000 037A top_isr equ     *
1523    0000037A 205F              movea.l (sp)+,a0        ; Save return addr
1524    0000037C 285F              movea.l (sp)+,a4        ; Get sc_subtabletype
1525    0000037E 4850              pea     (a0)            ; Replace ret addr
1526
1527    00000380 266C 0020         movea.l C_ADR(a4),a3    ; Get card base addr
1528
1529    00000384 528B              addq.l  #1,a3           ; Now a3.l = OUR base address of card
1530
1531                        **********  TRY SECTION: RECOVER IS AT END OF ISR
1532
1533    00000386 48E7 0018         movem.l a3/a4,-(sp)
1534    0000038A 2F2D FFF6         move.l  RCVR_BLK(a5),-(sp)
1535    0000038E 2F0E              move.l  a6,-(sp)
1536    00000390 487A 00CE         pea     recover_section
1537    00000394 2B4F FFF6         move.l  sp,RCVR_BLK(a5)
1538
1539                        **********  END OF 'TRY' keyword
1540
1541    00000398 396C 003C         move.w  term_and_mode(a4),int_savespace(a4)
                 0056
1542    0000039E 4CEC 003F         movem.l term_and_mode+2(a4),d0-d5
                 003E
```

```
1543    000003A4 48EC 003F          movem.l d0-d5,int_savespace+2(a4)
                 0068
1544
1545    000003AA 196B 4000          move.b  INT_COND(a3),intbits(a4) ;Get all the interrupt condition bits
                 003A
1546
1547                        *   From now on, each handler routine checks for its particular interrupting
1548                        *   condition and then jumps to the next.  This is done primarily for speed.
```



```
1550                        ********************************************************************************
1551                        *                                                                              *
1552                        *       routine remcont_reset_isr:      Takes care of communicating a          *
1553                        *               ==================      card's remote-cont-ol-reset to the     *
1554                        *                                       operating system                       *
1555                        *                                                                              *
1556                        ********************************************************************************
1557
1558             0000 03B0 remcont_reset_isr equ *
1559    000003B0 082C 0004          btst    #RC_reset_int,intbits(a4) ; Test for bit #4 for this condition
                 003A
1560    000003B6 6708               beq.s   error_isr
1561    000003B8 43EC 0072          lea     RCR_hook(a4),a1
1562    000003BC 6100 00DC          bsr     try_hook
1563
1564
1565
1566                        ********************************************************************************
1567                        *                                                                              *
1568                        *       routine error_isr:    Handles the communication of an error from the   *
1569                        *               ==========    interface card back to BASIC, or the hook.       *
1570                        *                                                                              *
1571                        *       At entry:                                                              *
1572                        *               a3.l =  card base address ($00xx0001)                          *
1573                        *               a4.l =  address of SCT: select_code_table                      *
1574                        *                                                                              *
1575                        *       Upon normal exit:                                                      *
1576                        *                                                                              *
1577                        *       The 'chk_err' routine is used also at powerup time.                    *
1578                        *                                                                              *
1579                        ********************************************************************************
1580
1581             0000 03C0 error_isr equ *
1582    000003C0 0822 0000          btst    #error_int,intbits(a4)  ; Test for bit #0 for this condition
                 003A
1583    000003C6 6724               beq.s   data_rx_isr
1584
1585    000003C8 6100 0004          bsr     chk_err
1586    000003CC 601E               bra.s   data_rx_isr
1587
1588    000003CE 4240       chk_err clr.w   d0
1589    000003D0 102B 400C          move.b  ERROR_CODE(a3),d0       ; fetch error number
1590    000003D4 6714               beq.s   errdone                 ; ignore ERROR_CODE=0
1591    000003D6 422B 400C          clr.b   ERROR_CODE(a3)
1592    000003DA D07C 012C          add.w   #300,d0                 ; add offset
1593
1594    000003DE 3940 0034          move.w  d0,ovrlaper(a4)
1595    000003E2 43EC 007A          lea     err_hook(a4),a1
1596    000003E6 6100 00B2          bsr     try_hook
1597
1598    000003EA 4E75       errdone rts
```

```
1600                        ********************************************************************************
1601                        *                                                                              *
1602                        *       routine data_rx_isr: Handles moving received data to the user's        *
1603                        *                            ===========  buffer for an inbound TRANSFER.    *
1604                        *                                                                              *
1605                        ********************************************************************************
1606
1607                0000 03EC data_rx_isr equ *
1608    000003EC 082C 0001          btst    #rx_int,intbits(a4)     ; Test for bit #1 for this condition
                    003A
1609    000003F2 6704               beq.s   data_tx_isr
1610    000003F4 6100 00CE          bsr     do_inxfr
1611
1612                        ********************************************************************************
1613                        *                                                                              *
1614                        *       routine data_tx_isr: Handles moving transmit data from the user's      *
1615                        *                            ===========  buffer to the card for an outbound TRANSFER.    *
1616                        *                                                                              *
1617                        ********************************************************************************
1618
1619                0000 03F8 data_tx_isr equ *
1620    000003F8 082C 0002          btst    #tx_int,intbits(a4)     ; Test for bit #2 for this condition
                    003A
1621    000003FE 6704               beq.s   ON_INTR_isr
1622    00000400 6100 0186          bsr     do_outxfr
1623
1624                        ********************************************************************************
1625                        *                                                                              *
1626                        *       routine ON_INTR_isr: Handles the communication of an ON INTR trigger   *
1627                        *                            ===========  from the interface card back to BASIC.    *
1628                        *                                                                              *
1629                        ********************************************************************************
1630
1631                0000 0404 ON_INTR_isr equ *
1632    00000404 082C 0003          btst    #ON_INTR_int,intbits(a4) ; Test for bit #3 for this condition
                    003A
1633    0000040A 6700 0016          beq     trace_isr
1634
1635    0000040E 08AC 0003          bclr    #ON_INTR_int,usr0mask(a4)
                    0036
1636    00000414 6704               beq.s   OIisr1          ; If already 0 don't send again
1637    00000416 6100 FE94          bsr     put_INTMASK
1638
1639    0000041A 43EC 0014 OIisr1   lea     USER_ISR(a4),a1
1640    0000041E 6100 0094          bsr     try_hock_P              (TM)    7/30/82 bug 158
1641
1642                0000 0422 trace_isr equ *
1643    00000422 082C 0005          btst    #trace_int,intbits(a4)  ; Test for bit #5 for this condition
                    003A
1644    00000428 6708               beq.s   bit_6_isr
1645    0000042A 43EC 0082          lea     trc_hock(a4),a1
1646    0000042E 6100 006A          bsr     try_hock
1647
1648                0000 0432 bit_6_isr equ *
1649    00000432 082C 0006          btst    #6,intbits(a4)          ; Test for bit #6 for this condition
                    003A
1650    00000438 6708               beq.s   bit_7_isr
```



```
1651    0000043A 43EC 008A          lea     bt6_hock(a4),a1
1652    0000043E 6100 005A          bsr     try_hock
```

```
1654                0000 0442 bit_7_isr equ *
1655    00000442 082C 0007      btst    #7,intbits(a4)          ; Test for bit #7 for this condition
                    004A
1656    00000448 6706           beq.s   end_isr
1657    0000044A 43EC 0092      lea     bt7_hook(a4),a1
1658    0000044E 6100 004A      bsr     try_hook
1659
1660
1661                        * -------------------------- End of the ISR -----------------------------------
1662
1663                0000 0452 end_isr equ     *
1664
1665                        ********** RECOVER SECTION FROM 'TRY' ABOVE *****
1666
1667    00000452 2B6F 0008      move.l  8(sp),RCVR_BLK(a5)
                    FFF6
1668    00000458 DFFC 0000      adda.l  #12,sp
                    000C
1669    0000045E 601E           bra.s   rcvdone
1670
1671                0000 0460 recover_section equ *          ; On escape, flag overlapped error
1672    00000460 2C5F           movea.l (sp)+,a6            ; to background
1673    00000462 2B5F FFF6      move.l  (sp)+,RCVR_BLK(a5)
1674
1675                        * Body of 'RECOVER' block:
1676    00000466 0C6D FFE6      cmpi.w  #IOE_ERROR,ESC_CODE(a5)
                    FFFE
1677    0000046C 6610           bne.s   rcvdone         ; Throw away non-I/O errors
1678    0000046E 102C 002D      move.b  IO_SC(a4),d0
1679    00000472 B02D FFBA      cmp.b   IOE_SC(a5),d0
1680    00000476 6606           bne.s   rcvdone
1681    00000478 396D FFBE      move.w  IOE_RSLT(a5),ovrlaper(a4)
                    0034
1682                        * That was it!
1683
1684    0000047E 4CDF 1800 rcvdone movem.l (sp)+,a3/a4
1685    00000482 6100 FCCE      bsr     eir
1686    00000486 39FC 0056      move.w  int_savespace(a4),term_and_mode(a4)
                    003C
1687    0000048C 4CEC 003F      movem.l int_savespace+2(a4),d0-d5
                    0058
1688    00000492 48EC 003F      movem.l d0-d5,term_and_mode+2(a4)
                    003E
1689    00000498 4E75           rts             ; Return from ISR
1690
1691
1692    0000049A 2011       try_hook move.l (a1),d0
1693    0000049C 6714           beq.s   hook1
1694    0000049E 48E7 0018      movem.l a3/a4,-(sp)
1695    000004A2 2040           movea.l d0,a0
1696    000004A4 2029 0004 HOOK4    move.l  4(a1),d0                (tm)    12/03/81
1697    000004A8 6702           beq.s   hook3                   (tm)    If there is static link - push it
1698    000004AA 2F00           move.l  d0,-(sp)                (tm)    ---------- said this is okay
1699    000004AC 4E90       hook3   jsr     (a0)
1700    000004AE 4CDF 1800 hook2    movem.l (sp)+,a3/a4
1701    000004B2 4E75       hook1   rts
1702
```



```
1703
1704    000004B4 2011       try_hook_P move.l (a1),d0               (TM)    7/30/82 bug 158
1705    000004B6 67FA           beq.s   hook1
1706    000004B8 48E7 0018      movem.l a3/a4,-(sp)
1707    000004BC 2040           movea.l d0,a0
1708    000004BE 2F29 0008      MOVE.L  8(A1),-(SP)             (TM)    7/30/82 bug 158
1709    000004C2 60E0           BRA     HOOK4                   (TM)    7/30/82 bug 158
```

```
1711                        ****************************************************
1712                        *                                                  *
1713                        *   routine do_inxfr:  Transfers data in until:    *
1714                        *                - User buffer filled;             *
1715                        *                - Card buffer empty; or           *
1716                        *                - Control block reached.          *
1717                        *                                                  *
1718                        *   At entry:                                      *
1719                        *       a3.l =   card base address ($00xx0001)     *
1720                        *       a4.l =   address of select_code_table      *
1721                        *                                                  *
1722                        ****************************************************
1723
1724              0000 04C4 do_inxfr equ    *
1725    000004C4 246C 0024          movea.l BUFI_OFF(a4),a2
1726    000004C8 6700 008A          beq     inxdone
1727
1728    000004CC 296A 0010          move.l  TCNT_OFF(a2),data_number(a4)
                 0042
1729    000004D2 6700 0090          beq     inxdn1
1730    000004D6 296A 0020          move.l  TFIL_OFF(a2),data_address(a4)
                 003E
1731
1732    000004DC 6100 FBD6 inxfr1   bsr     find_RXBUF      ; Set up a2.l = buffer descriptor record
1733                        *                                                   base address
1734    000004E0 6100 030A inxfr4   bsr     RX_stuff_avail  ; See if buffer is empty
1735    000004E4 4A00               tst.b   d0              ; If so, just sit here & wait
1736    000004E6 6700 008C          beq     inxexit
1737
1738    000004EA 6100 035E          bsr     ctrlblknext     ; If a control block is next, then
1739    000004EE 4A00               tst.b   d0
1740    000004F0 663E               bne.s   inxfr3
1741
1742    000004F2 4AAC 0042          tst.l   data_number(a4) ; see if chars to transfer
1743    000004F6 675C               beq.s   inxdone         ; yes, go do it
1744
1745    000004F8 226C 0024          movea.l BUFI_OFF(a4),a1
1746    000004FC 3029 000E          move.w  TCHR_OFF(a1),d0 ; Check for term char desired
1747    00000500 6C06               bge.s   inxfr5          ; Yes - goto slow section
1748
1749    00000502 6100 0274          bsr     getchars        ; move some data
1750                        *                               ; & go back to check for ctrl blk
1751    00000506 60D4               bra.s   inxfr1
1752
1753              0000 0508 inxfr5  equ     *       ; SLOW ENTER - search for char
1754    00000508 1F00               move.b  d0,-(sp)        ; Save search char
1755    0000050A 2F2C 0042          move.l  data_number(a4),-(sp)
1756    0000050E 297C 0000          move.l  #1,data_number(a4)
                 0001 0042
1757    00000516 6100 0260          bsr     getchars
1758    0000051A 295F 0042          move.l  (sp)+,data_number(a4)
1759    0000051E 53AC 0042          subq.l  #1,data_number(a4)
1760    00000522 101F               move.b  (sp)+,d0        ; Check for term chr
1761    00000524 206C 003E          movea.l data_address(a4),a0
1762    00000528 B028 FFFF          cmp.b   -1(a0),d0       ; If equal, exit
1763    0000052C 6726               beq.s   inxdone
1764    0000052E 60AC               bra.s   inxfr1
```



```
1765
1766    00000530 6100 020C inxfr3   bsr     getctrlblk      ; Get it, and check for the special
1767    00000534 0C2C 00FF          cmpi.b  #255,term(a4)   ; case TERM=255
                 003C
1768    0000053A 6608               bne.s   inxfr2
1769    0000053C 196C 003D          move.b  mode(a4),which_RXbuf(a4) ;If so, do the buffer switch
                 0037
1770    00000542 6098               bra.s   inxfr1          ; And go back for more
1771
1772    00000544 396C 003C inxfr2   move.w  term_and_mode(a4),last_enter_term(a4)
                 0038
1773                        *                               ; Otherwise save the control block & leave
1774    0000054A 246C 0024          movea.l BUFI_OFF(a4),a2
1775    0000054E 4A2A 000B          tst.b   TEND_OFF(a2)    ; If EOI term bit set tghen
1776    00000552 6788               beq.s   inxfr1          ; leave else ignore
1777
1778    00000554 246C 0024 inxdone  movea.l BUFI_OFF(a4),a2
1779    00000558 256C 003E          move.l  data_address(a4),TFIL_OFF(a2)
                 0020
1780    0000055E 256C 0042          move.l  data_number(a4),TCNT_OFF(a2)
                 0010
1781    00000564 6100 00AC inxdn1   bsr     inxfr_done
1782    00000568 200A               move.l  a2,d0
1783    0000056A 671A               beq.s   inxex1
1784    0000056C 43EA 0024          lea     T_PR_OFF(a2),a1
1785    00000570 6000 FF42          bra     try_hook_P                      (TM)    7/30/82 bug 158
1786
1787              0000 0574 inxexit equ     *
1788    00000574 246C 0024          movea.l BUFI_OFF(a4),a2
1789    00000578 256C 003E          move.l  data_address(a4),TFIL_OFF(a2)
                 0020
1790    0000057E 256C 0042          move.l  data_number(a4),TCNT_OFF(a2)
                 0010
1791    00000584 67DE               beq.s   inxdn1                          BUG 1249   TM 01/08/82
1792    00000586 4E75       inxex1  rts
```

```
1794                        ************************************************
1795                        *                                              *
1796                        *   routine do_outxfr:  Transfers data out til: *
1797                        *                - User buffer emptied; or     *
1798                        *                - Card buffer filled.         *
1799                        *                                              *
1800                        *   At entry:                                  *
1801                        *        a3.l =  card base address ($00xx0001) *
1802                        *        a4.l =  address of select_code_table  *
1803                        *                                              *
1804                        *   During execution:                          *
1805                        *        a2.l =  address of xfr record         *
1806                        *                                              *
1807                        ************************************************
1808
1809             0000 0588 do_outxfr equ   *
1810    00000588 246C 0028         movea.l BUFO_OFF(a4),a2
1811    0000058C 200A              move.l  a2,d0
1812    0000058E 67RE              beq.s   outxex1
1813
1814    00000590 29RA 0010         move.l  TCNT_OFF(a2),data_number(a4)
                0042
1815    00000596 6730              beq.s   outxd0
1816    00000598 29RA 001C         move.l  TEMP_OFF(a2),data_address(a4)
                003E
1817
1818    0000059E 6100 FB10         bsr     find_TXBUF      ; a2 = buff descr rec addr
1819
1820    000005A2 2F2C 0042 outx1   move.l  data_number(a4),-(sp)
1821    000005A6 6100 055C         bsr     putchars        ; Send some chars
1822    000005AA 201F              move.l  (sp)+,d0
1823    000005AC B0AC 0042         cmp.l   data_number(a4),d0
1824    000005B0 672C              beq.s   outxit          ; If no chars transferred
1825    000005B2 4AAC 0042         tst.l   data_number(a4) ; then exit
1826    000005B6 66EA              bne.s   outx1           ; Check buffer empty
1827
1828    000005B8 246C 0028 outxdun movea.l BUFO_OFF(a4),a2
1829    000005BC 256C 003E         move.l  data_address(a4),TEMP_OFF(a2)
                001C
1830    000005C2 256C 0042         move.l  data_number(a4),TCNT_OFF(a2)
                0010
1831    000005C8 246C 0028 outxd0  movea.l BUFO_OFF(a4),a2
1832    000005CC 4A2A 000B         tst.b   TEND_OFF(a2)    ; Check for END termination
1833    000005D0 670C              beq.s   outxdn1
1834    000005D2 6100 FADC         bsr     find_TXBUF      ; a2 = buff descr rec addr
1835    000005D6 6100 0660         bsr     try_sending_EOF
1836    000005DA 4A00              tst.b   d0              ; If couldn't send it then
1837    000005DC 6720              beq.s   outxex1         ; return, wait for next interrupt
1838
1839    000005DE 6100 0020 outxdn1 bsr     outxfr_done
1840    000005E2 200A              move.l  a2,d0
1841    000005E4 6718              beq.s   outxex1
1842    000005E6 43EA 0024         lea     T_PR_OFF(a2),a1
1843    000005EA 6000 FEC8         bra     try_hook_P                  (TM)    7/30/82 bug 158
1844
1845    000005EE 24AC 0028 outxit  movea.l BUFO_OFF(a4),a2
```



```
1846    000005F2 256C 003E         move.l  data_address(a4),TEMP_OFF(a2)
                001C
1847    000005F8 256C 0042         move.l  data_number(a4),TCNT_OFF(a2)
                0010
1848    000005FE 4E75      outxex1 rts
```

```
1850                        ************************************************
1851                        *                                              *
1852                        *   routine outxfr_done:  Terminates the       *
1853                        *                 outbound transfer, if any.   *
1854                        *                                              *
1855                        *   routine inxfr_done:  Terminates the inbound *
1856                        *                 transfer, if any.            *
1857                        *                                              *
1858                        *   At entry:                                  *
1859                        *       a3.l =  card base address ($00xx0001)  *
1860                        *       a4.l =  address of select_code_table   *
1861                        *                                              *
1862                        ************************************************
1863
1864                0000 0600 outxfr_done equ *
1865    00000600 08AC 0002          bclr    #tx_int,usr0mask(a4)
                    0036
1866    00000606 6704               beq.s   outxd1          ; If already 0 don't send again
1867    00000608 6100 FCA2          bsr     put_INTMASK
1868    0000060C 41EC 0028 outxd1   lea     BUF0_OFF(a4),a0
1869    00000610 6010               bra.s   xfrdun
1870
1871
1872                0000 0612 inxfr_done equ *
1873    00000612 08AC 0001          bclr    #rx_int,usr0mask(a4)
                    0036
1874    00000618 6704               beq.s   inxd1           ; If already 0 don't send again
1875    0000061A 6100 FC90          bsr     put_INTMASK
1876    0000061E 41EC 0024 inxd1    lea     BUF1_OFF(a4),a0
1877
1878    00000622 2450       xfrdun  movea.l (a0),a2
1879    00000624 200A               move.l  a2,d0
1880    00000626 670C               beq.s   rtsin
1881    00000628 157C 00FF          move.b  #255,T_SC_OFF(a2)
                    0005
1882    0000062E 422A 0007          clr.b   TACT_OFF(a2)
1883    00000632 4290               clr.l   (a0)
1884    00000634 4E75       rtsin   rts
```



```
1886                        ************************************************
1887                        *                                              *
1888                        *   routine wait_outxfrdone: waits until an    *
1889                        *                 outbound transfer is complete*
1890                        *                 (if any).  Also has timeout  *
1891                        *                 escape.                      *
1892                        *                                              *
1893                        *   routine wait_inxfrdone: waits until an     *
1894                        *                 inbound transfer is complete *
1895                        *                 (if any).  Also has timeout  *
1896                        *                 escape.                      *
1897                        *                                              *
1898                        *   At entry:                                  *
1899                        *       a3.l =  card base address ($00xx0001)  *
1900                        *       a4.l =  address of select_code_table   *
1901                        *                                              *
1902                        *   This bashes nothing.                       *
1903                        *                                              *
1904                        *   This routine may escape!                   *
1905                        *                                              *
1906                        *   MODIFIED                12/02/81           *
1907                        *            from 12/01/81 code to 11/23/81    *
1908                        *                                              *
1909                        ************************************************
1910
1911                0000 0636 wait_outxfrdone equ *
1912    00000636 48E7 A080          movem.l d0/d2/a0,-(sp)
1913    0000063A 206C 0028          movea.l BUF0_OFF(a4),a0
1914    0000063E 6008               bra.s   wait
1915
1916                0000 0640 wait_inxfrdone equ *
1917    00000640 48E7 A080          movem.l d0/d2/a0,-(sp)
1918    00000644 206C 0028          movea.l BUF0_OFF(a4),a0
1919
1920    00000648 2408       wait    move.l  a0,d2           ; 0=no xfr block
1921    0000064A 6730               beq.s   waitdun
1922    0000064C 0C28 0005          cmpi.b  #5,TUSR_OFF(A0) ; >4 = interrupt
                    0009
1923    00000652 6D28               blt.s   waitdun
1924    00000654 242C 002E          move.l  timeout(a4),d2
1925    00000658 0838 0001          btst    #timer_present,sysflag2   is timer available ? tttt JS 8/11/83
                    FEEA
1926    0000065E 6722               beq.s   wait_timer              if so then use it    tttt JS 8/11/83
1927
1928    00000660 203C 0000 wait1    move.l  #256,d0         [UNCALIBRATED] - guess based upon check_tfr loop
                    0100
1929    00000666 4A28 0007 wait2    tst.b   TACT_OFF(a0)    ; 0=inactive            (tm)  tst.l -> tst.b
1930    0000066A 6710               beq.s   waitdun
1931
1932    0000066C 5380               subq.l  #1,d0           ; Timeout computation
1933    0000066E 66F6               bne.s   wait2
1934    00000670 4A82               tst.l   d2
1935    00000672 67EC               beq.s   wait1
1936    00000674 5382               subq.l  #1,d2       bug fix -- was d0               tttt JS 8/11/83
1937    00000676 66E8               bne.s   wait1
1938    00000678 6000 FCE2          bra     time_err
1939
```

```
1940    0000067C 4CDF 0105 waitdun movem.l (sp)+,d0/d2/a0
1941    00000680 4E75              rts
1942
1943              0000 0682 wait_timer equ *
1944    00000682 4A82              tst.l   d2              check for infinite timeout    tttt JS 8/11/83
1945    00000684 67DA              beq     wait1           use loops if infinite         tttt JS 8/11/83
1946    00000686 1F3C 0001         move.b  #1,-(sp)           set up timer record        tttt JS 8/11/83
1947    0000068A 2F02              move.l  d2,-(sp)           d2 has ms to wait          tttt JS 8/11/83
1948    0000068C 4A28 0007 wait3   tst.b   tact_off(a0)       xfr done?                  tttt JS 8/11/83
1949    00000690 670E              beq.s   wait4              if so then exit loop       tttt JS 8/11/83
1950    00000692 4857              pea     (sp)               push ptr to time rec       tttt JS 8/11/83
1951    00000694 4EB9 0000         jsr     check_timer        and check the timer        tttt JS 8/11/83
                 0000
1952    0000069A 6AF0              bpl     wait3              if no timeout, keep trying tttt JS 8/11/83
1953    0000069C 6000 FCBE         bra     time_err           timeout -- go escape       tttt JS 8/11/83
1954    000006A0 5C4F      wait4   addq    #6,sp              normal exit -- clean stack tttt JS 8/11/83
1955    000006A2 60D8              bra     waitdun            and return                 tttt JS 8/11/83
1956
```



```
1958                    ************************************************
1959                    *                                              *
1960                    *   procedure START_TRANSFER_IN (              *
1961                    *               var SCT: select_code_table );  *
1962                    *                                              *
1963                    *       This starts the card doing a transfer. *
1964                    *       The calling code must have already     *
1965                    *       linked the transfer block in to the    *
1966                    *       select_code_table structure.           *
1967                    *                                              *
1968                    *   During use:                                *
1969                    *       a3.l =  card base address ($00xx0001)  *
1970                    *       a4.l =  address of select_code_table   *
1971                    *                                              *
1972                    ************************************************
1973
1974              0000 06A4 START_TRANSFER_IN equ *
1975    000006A4 205F              movea.l (sp)+,a0
1976    000006A6 285F              movea.l (sp)+,a4        ; SCT
1977    000006A8 4850              pea     (a0)
1978    000006AA 266C 0020         movea.l c_adr(a4),a3
1979    000006AE 528B              addq.l  #1,a3
1980
1981    000006B0 4E4B              trap    #11             get into supervisor mode          scs
1982                    * scs      move    sr,-(sp)        ; Funny code to disable
1983    000006B2 46FC 0070         move    SR_image(a4),sr ; interrupts
1984    000006B6 08EC 0001         bset    #rx_int,usr0mask(a4)
                 0006
1985    000006BC 6100 FBEE         bsr     put_INTMASK
1986    000006C0 6100 FE02         bsr     do_Inxfr
1987    000006C4 46DF              move    (sp)+,sr
1988    000006C6 6000 FC74         bra     check_ov_error  ; Will enable ints
```

```
1990                    ************************************************
1991                    *                                              *
1992                    *   procedure START_TRANSFER_OUT (             *
1993                    *               var SCT: select_code_table );  *
1994                    *                                              *
1995                    *       This starts the card doing a transfer. *
1996                    *       The calling code must have already     *
1997                    *       linked the transfer block in to the    *
1998                    *       select_code_table structure.           *
1999                    *                                              *
2000                    *   During use:                                *
2001                    *       a3.l =  card base address ($00xx0001)  *
2002                    *       a4.l =  address of select_code_table   *
2003                    *                                              *
2004                    ************************************************
2005
2006            0000 06CA START_TRANSFER_OUT equ *
2007    000006CA 205F           movea.l (sp)+,a0
2008    000006CC 285F           movea.l (sp)+,a4        ; SCT
2009    000006CE 4850           pea     (a0)
2010    000006D0 266C 0020      movea.l c_adr(a4),a3
2011    000006D4 528B           addq.l  #1,a3
2012
2013    000006D6 4E4B           trap    #11                                                     scs
2014                    * scs   move    sr,-(sp)        ; Funny code to disable
2015    000006D8 46EC 0070      move    SR_image(a4),sr ; interrupts
2016    000006DC 08EC 0002      bset    #tx_int,usr0mask(a4)
                0036
2017    000006E2 6100 FBC8      bsr     put_INTMASK
2018    000006E6 6100 FEA0      bsr     do_outxfr
2019    000006EA 46DF           move    (sp)+,sr
2020    000006EC 6000 FC4E      bra     check_ov_error  ; Will enable ints
2021
```



```
2024
2025
2026                    *       ****    *   *   ****    *   *   *****   *****   *****   ****
2027                    *       *   *   *   *   *   *   *   *   *       *       *       *   *
2028                    *       *   *    * *    *   *   *   *   *       *       *       *   *
2029                    *       ****      *     ****    *   *   *****   *****   *****   ****
2030                    *       *  *      * *    *   *   *   *   *       *       *       *  *
2031                    *       *   *   *   *   *   *   *   *   *       *       *       *   *
2032                    *       *   *   *   *   ****     ***    *       *       *****   *   *
2033
2034
2035
2036
2037
2038                    ************************************************************************
2039                    *                                                                      *
2040                    *       routine set_RXBUF_a6:   Routine to set a6 as the base address  *
2041                    *                               pointer to whichever Rx buffer is being used.  *
2042                    *                                                                      *
2043                    *       At entry:                                                      *
2044                    *               a2.l =  data buffer base address (shifted, +1+selectcode)      *
2045                    *               a3.l =  card base address ($00xx0001)                  *
2046                    *               a4.l =  pointer to sc_subtabletype structure           *
2047                    *                                                                      *
2048                    *       Upon exit:                                                     *
2049                    *               a6.l =  Base address of DATA_BUFFERS[WHICH_RXBUF]      *
2050                    *               This also bashes d0.                                   *
2051                    *                                                                      *
2052                    ************************************************************************
2053
2054            0000 06F0 set_RXBUF_a6 equ *
2055    000006F0 4280           clr.l   d0                      ; Setup d0.l=offset
2056    000006F2 102C 0037      move.b  which_RXbuf(a4),d0      ; to which Rx buffer
2057    000006F6 E980           asl.l   #4,d0                   ; being used
2058    000006F8 4DF2 0010      lea     DATA_AREA(a2,d0),a6
2059    000006FC 4E75           rts
```

```
2061                    ********************************************************************************
2062                    *                                                                              *
2063                    *       routine RX_BUFF_bytes:  Function which returns the number of           *
2064                    *               =============   characters until the first control block in    *
2065                    *                               the Receive Buffer.  If there are no control   *
2066                    *                               blocks, this just returns the number of        *
2067                    *                               characters in the buffer.  This only works     *
2068                    *                               on the current Rx data buffer, and does not    *
2069                    *                               extract TERM=255 control blocks!               *
2070                    *                                                                              *
2071                    *       At entry:                                                              *
2072                    *               a2.l =  data buffer base address (shifted, +1+selectcode)      *
2073                    *               a3.l =  card base address ($00xx0001)                          *
2074                    *               a4.l =  pointer to sc_subtabletype structure                   *
2075                    *                                                                              *
2076                    *       Upon exit:                                                             *
2077                    *               d0.l =  Number of characters.                                  *
2078                    *               a1, d4 and d5 are left with values from find_RX_DATA.          *
2079                    *               This also bashes a0, d1 and d2.                                *
2080                    *                                                                              *
2081                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2082                    *                                                                              *
2083                    *       This routine calls gain_access, release_access, and find_RX_DATA.      *
2084                    *                                                                              *
2085                    ********************************************************************************
2086
2087            0000 06FE RX_BUFF_bytes equ *
2088    000006FE 6100 F9DA          bsr     find_RX_DATA            ; Setup a1 = data buffer base addr
2089                    *                                                       d4 = end of data buffer addr
2090                    *                                                       d5 = RXDATABUFF_SIZE
2091    00000702 2F0E               move.l  a6,-(sp)
2092    00000704 61EA               bsr     set_RXBUF_a6
2093
2094    00000706 4280               clr.l   d0                      ; Get garbage out of top of d0, d1
2095    00000708 4281               clr.l   d1
2096    0000070A 030A 000C          movep.w CTRL_AREA+EMPTY(a2),d1  ; Fetch pointers (bytes in wrong order)
2097    0000070E 6100 F91A          bsr     gain_access             ; Need access to FILL pointers
2098    00000712 010E 0008          movep.w FILL(a6),d0
2099    00000716 030A 0008          movep.w CTRL_AREA+FILL(a2),d2
2100    0000071A 6100 F96A          bsr     release_access
2101    0000071E B242               cmp.w   d2,d1                   ; If the two ctrl block pointers are not
2102    00000720 670A               beq.s   RBb1                    ; equal, then we want to use the pointer
2103    00000722 EE59               ror.w   #7,d1                   ; field from the next control block to
2104    00000724 D28B               add.l   a3,d1                   ; indicate how much data may be removed
2105    00000726 2041               movea.l d1,a0
2106    00000728 0108 0000          movep.w POINTER(a0),d0          ; --- Use it as the "FILL" pointer
2107
2108    0000072C E058       RBb1    ror.w   #8,d0                   ; Switch bytes for FILL
2109    0000072E 030E 000C          movep.w EMPTY(a6),d1            ; and get EMPTY and switch bytes
2110    00000732 E059               ror.w   #8,d1                   ; d0="FILL", d1=EMPTY
2111
2112    00000734 9041               sub.w   d1,d0                   ; Compute d0 := FILL-EMPTY
2113    00000736 6C02               bge.s   RBb2
2114    00000738 D045               add.w   d5,d0                   ; If negative, add data buffer size
2115    0000073A 2C5F       RBb2    movea.l (sp)+,a6
2116    0000073C 4E75               rts     ; Now d0 = ("FILL"-EMPTY) mod SIZE --- of data buffer
```



```
2118                    ********************************************************************************
2119                    *                                                                              *
2120                    *       routine getctrlblk:     Routine which gets a control block from the    *
2121                    *               ==========      Receive buffer.  It must have already been     *
2122                    *                               determined that there is a control block at    *
2123                    *                               the front of the buffer, since this routine    *
2124                    *                               does NOT check for that condition.  The TERM   *
2125                    *                               and MODE fields of the removed block are left  *
2126                    *                               in the appropriate (.term and .mode) in the    *
2127                    *                               sc_subtabletype structure.                     *
2128                    *                                                                              *
2129                    *       At entry:                                                              *
2130                    *               a2.l =  RX buffer record base address from find_RXBUF          *
2131                    *               a3.l =  card base address ($00xx0001)                          *
2132                    *               a4.l =  pointer to sc_subtabletype structure                   *
2133                    *                                                                              *
2134                    *       Upon exit:                                                             *
2135                    *               sc_subtabletype.term =  TERM field of control block (8 bits)   *
2136                    *               sc_subtabletype.mode =  MODE field of control block (8 bits)   *
2137                    *               a1, d4 and d5 are left with the values from find_CTRL_AREA.    *
2138                    *               This bashes d0, d2, and a0.                                    *
2139                    *                                                                              *
2140                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2141                    *                                                                              *
2142                    *       This routine calls gain_access, release_access, and find_CTRL_AREA.    *
2143                    *                                                                              *
2144                    ********************************************************************************
2145
2146            0000 073E getctrlblk equ  *
2147    0000073E 6100 F9B4          bsr     find_CTRL_AREA          ; Setup a1 = ctrl buffer base addr
2148                    *                                                       d4 = end of ctrl buffer addr
2149                    *                                                       d5 = TRCTRLBUFF_SIZE
2150
2151    00000742 4280               clr.l   d0                      ; Clear top of d0
2152    00000744 010A 000C          movep.w CTRL_AREA+EMPTY(a2),d0  ; Get control buffer EMPTY pointer
2153    00000748 EE58               ror.w   #7,d0                   ; Now make it into a 68000 pointer
2154    0000074A D08B               add.l   a3,d0
2155    0000074C 2040               movea.l d0,a0                   ; Move to a0 so we can use it
2156
2157    0000074E 1968 0004          move.b  TERMFIELD(a0),term(a4)  ; Store term & mode fields
            003C
2158    00000754 1968 0006          move.b  MODEFIELD(a0),mode(a4)
            003D
2159
2160    0000075A D07C 0008          add.w   #CTRLBLKSIZE,d0         ; Bump pointer by control block size
2161    0000075E B840               cmp.w   d0,d4                   ; and check for wraparound.
2162    00000760 6602               bne.s   gcb1
2163    00000762 3009               move.w  a1,d0                   ; If so, set to front of buffer
2164    00000764 0880 0000 gcb1     bclr    #0,d0                   ; Make it into a Z80
2165    00000768 EF58               rol.w   #7,d0                   ; type pointer with bytes reversed
2166    0000076A 6100 F8BE          bsr     gain_access             ; Now store the updated EMPTY pointer
2167    0000076E 018A 000C          movep.w d0,CTRL_AREA+EMPTY(a2)
2168    00000772 6100 F912          bsr     release_access
2169    00000776 4E75               rts     ;<<<CAN'T COMBINE WITH ABOVE!!!!!
```

```
2171                    ********************************************************************************
2172                    *                                                                              *
2173                    *       routine getchars:       Routine which takes characters from the        *
2174                    *               ========        Receive buffer and puts them in the area       *
2175                    *                               pointed to by sc_subtabletype.data_address     *
2176                    *                               and sized by sc_subtabletype.data_number.      *
2177                    *                               The number of characters                       *
2178                    *                               actually transfered is the minimum of:         *
2179                    *                               (1) the number of characters available before  *
2180                    *                               the first Receive buffer control block;        *
2181                    *                               (2) sc_subtabletype.data_number;               *
2182                    *                               and (3) the number of characters               *
2183                    *                               available until the Receive buffer wraparound  *
2184                    *                               point.  THIS NUMBER MAY BE ZERO!               *
2185                    *                               This alters data_address and data_number to    *
2186                    *                               reflect where to start going next time this    *
2187                    *                               is called.  The criteria for ending the        *
2188                    *                               transfer at a higher level must be determined  *
2189                    *                               by data_number, RX_stuff_avail and             *
2190                    *                               ctrl_blk_next/getctrlblk.                      *
2191                    *                                                                              *
2192                    *       At entry:                                                              *
2193                    *               a2.l =  RX buffer record base address from find_RXBUF          *
2194                    *               a3.l =  card base address ($00xx0001)                          *
2195                    *               a4.l =  pointer to sc_subtabletype structure                   *
2196                    *                                                                              *
2197                    *       Upon exit:                                                             *
2198                    *               data_address and data_number are updated, plus the EMPTY       *
2199                    *               pointer in the card's Receive data buffer.                     *
2200                    *               In sc_subtabletype last_enter_term and last_enter_mode are     *
2201                    *               zeroed if any data is moved.                                   *
2202                    *               a1 and d4 are left with the values from find_RX_DATA.          *
2203                    *               This bashes d0, d1, d2, d3, d4, d5, a0, and a1.                *
2204                    *                                                                              *
2205                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2206                    *                                                                              *
2207                    *       This routine calls gain_access, release_access, and RX_BUFF_bytes.     *
2208                    *                                                                              *
2209                    ********************************************************************************
2210
2211               0000 0778 getchars equ    *
2212    00000778 6184          bsr     RX_BUFF_bytes           ; Setup a1 = data buffer base addr
2213                    *                                                 d3 = offset to which Rxbuff used
2214                    *                                                 d4 = end of data buffer addr
2215                    *                                                 d5 = RXDATABUFF_SIZE
2216    0000077A 2600          move.l  d0,d3                   ; d3.l = available characters
2217
2218    0000077C 48E7 0022     movem.l a2/a6,-(sp)             ; Saved for local use
2219    00000780 6100 FF6E     bsr     set_RXBUF_a6
2220
2221    00000784 4280          clr.l   d0
2222    00000786 010E 000C     movep.w EMPTY(a6),d0            ; Get RXDATABUFF_EMPTY and make
2223    0000078A EE58          ror.w   #7,d0                   ; it into a 68000 pointer
2224    0000078C D08B          add.l   a3,d0
2225    0000078E 2440          movea.l d0,a2                   ; Save EMPTY for later!
2226
2227    00000790 9084          sub.l   d4,d0
```



```
2228    00000792 4480          neg.l   d0                      ; d0 = wraparound address - EMPTY
2229    00000794 C0BC 0000     and.l   #$0000FFFE,d0
                 FFFE
2230    0000079A E258          ror.w   #1,d0                   ; d0.l = number of bytes till wraparound
2231
2232    0000079C B680          cmp.l   d0,d3                   ; If d0>d3 then set d0 := d3
2233    0000079E 6E02          bgt.s   gc1
2234    000007A0 2003          move.l  d3,d0
2235
2236    000007A2 242C 0042 gc1 move.l  data_number(a4),d2      ; Fetch number of positions available
2237    000007A6 B480          cmp.l   d0,d2                   ; If d0>d2 then set d0 := d2
2238    000007A8 6E02          bgt.s   gc2
2239    000007AA 2002          move.l  d2,d0
2240
2241    000007AC 2600      gc2 move.l  d0,d3                   ; d3.l saves number of chars actually
2242                       *                                   ; transferred below
2243    000007AE 6736          beq.s   gcdone                  ; If zero, no work to be done
2244    000007B0 426C 0038     clr.w   last_enter_term(a4)     ; This also clears last_enter_mode.
2245    000007B4 5340          subq.w  #1,d0                   ; Make offset correct for dbf instr.
2246    000007B6 206C 003E     movea.l data_address(a4),a0     ; Get character pointer into a0
2247
2248    000007BA 10D2      gcloop move.b  (a2),(a0)+           ; Transfer a character & bump dest ptr
2249    000007BC 544A          addq.w  #2,a2                   ; Bump source pointer (odd bytes)
2250    000007BE 51C8 FFFA     dbf     d0,gcloop               ; Then decrement d0 & loop
2251
2252    000007C2 9483          sub.l   d3,d2                   ; Decrement datacnt by # bytes
2253    000007C4 2942 0042     move.l  d2,data_number(a4)      ; Now store adjusted address and
2254    000007C8 2948 003E     move.l  a0,data_address(a4)     ; number fields
2255
2256    000007CC 220A          move.l  a2,d1                   ; Store pointer for computations
2257    000007CE B88A          cmp.l   a2,d4                   ; Now check to see if EMPTY was moved
2258    000007D0 6602          bne.s   gc3                     ; past end of buffer.  If so, set to
2259    000007D2 2209          move.l  a1,d1                   ; the front of the buffer.
2260    000007D4 0881 0000 gc3 bclr    #0,d1                   ; Fix up the 68000 pointer to be the
2261    000007D8 EF59          rol.w   #7,d1                   ; card's type of pointer
2262    000007DA 6100 F84E     bsr     gain_access
2263    000007DE 038E 000C     movep.w d1,EMPTY(a6)            ; Remember d1 = card's EMPTY pointer.
2264    000007E2 6100 F8A2     bsr     release_access
2265    000007E6 4CDF 4400 gcdone movem.l (sp)+,a6/a2
2266    000007EA 4E75          rts
```

```
2268                    ********************************************************************************
2269                    *                                                                              *
2270                    *       routine RX_stuff_avail: Routine which determines whether there is      *
2271                    *               ===============  ANYTHING (data or control blocks) in the      *
2272                    *                                Receive buffer.  This consumes any TERM=255   *
2273                    *                                control blocks before returning the function. *
2274                    *                                                                              *
2275                    *       At entry:                                                              *
2276                    *               a2.l =  RX buffer record base address from find_RXBUF          *
2277                    *               a3.l =  card base address ($00xx0001)                          *
2278                    *               a4.l =  pointer to sc_subtabletype structure                   *
2279                    *                                                                              *
2280                    *       Upon exit:                                                             *
2281                    *               d0.l =  $00 if buffer is empty,                                *
2282                    *                       $01 if ctrl buffer is empty and data buffer is not,    *
2283                    *                       $02 if data buffer is empty and ctrl buffer is not,    *
2284                    *                       $03 if both data and ctrl buffers are not empty.       *
2285                    *               a1 and d4 are left with the values from find_RX_DATA.          *
2286                    *               This bashes d0, d1, d2, d3, d4, d5, a0 and a1.                 *
2287                    *                                                                              *
2288                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2289                    *                                                                              *
2290                    *       This routine calls gain_access and release_access.                     *
2291                    *                                                                              *
2292                    ********************************************************************************
2293
2294             0000 07EC RX_stuff_avail equ *
2295    000007EC 6100 F8EC         bsr     find_RX_DATA            ; Setup a1 = data buffer base addr
2296                    *                                                  d4 = end of data buffer addr
2297                    *                                                  d5 = RXDATABUFF_SIZE
2298    000007F0 2F0E               move.l  a6,-(sp)
2299    000007F2 6100 FEFC          bsr     set_RXBUF_a6
2300
2301    000007F6 6100 F832          bsr     gain_access
2302    000007FA 0C2E 0008          movep.w FILL(a6),d3             ; Fetch FILL & EMPTY (bytes reversed but
2303    000007FE 0C2A 0008          movep.w CTRL_AREA+FILL(a2),d1
2304    00000802 6100 F882          bsr     release_access          ; we're just checking equality)
2305    00000806 4282               clr.l   d2
2306    00000808 4280               clr.l   d0
2307    0000080A 0C2A 000C          movep.w CTRL_AREA+EMPTY(a2),d2
2308    0000080E B441               cmp.w   d1,d2                   ; Compare ctrl buff FILL & EMPTY
2309    00000810 660E               bne.s   setbit1                 ; If not equal, then set bit 1
2310    00000812 0C2E 000C chkdata  movep.w EMPTY(a6),d2
2311    00000816 B443               cmp.w   d3,d2                   ; Compare data buff FILL & EMPTY
2312    00000818 6702               beq.s   return
2313    0000081A 5200               addq.b  #1,d0                   ; And set bit 0 if not equal
2314    0000081C 2C5F       return  movea.l (sp)+,a6
2315    0000081E 4E75               rts
2316
2317    00000820 5400       setbit1 addq.b  #2,d0                   ; Set "ctrl not empty" bit
2318    00000822 EE5A               ror.w   #7,d2                   ; Something in control buffer - see if
2319                    *                                           ; this control block is at the head of
2320    00000824 D48B               add.l   a3,d2                   ; the queue (bytes reversed!)
2321    00000826 2C42               movea.l d2,a0
2322    00000828 0C28 0000          movep.w POINTER(a0),d1
2323    0000082C 0C2E 000C          movep.w EMPTY(a6),d2
2324    00000830 B442               cmp.w   d2,d1                   ; if POINTER field<>DATABUFF_EMPTY
```



```
2325    00000832 66DE               bne.s   chkdata                 ;   then go check data buff
2326    00000834 0C28 00FF          cmpi.b  #255,TERMFIELD(a0)      ; else if it's a TERM=255 control block
                 0004
2327    0000083A 66D6               bne.s   chkdata                 ; No, go back and check data buff
2328
2329    0000083C 6100 FF00          bsr     getctrlblk              ; Otherwise consume the control block
2330    00000840 196C 003D          move.b  mode(a4),which_RXbuf(a4) ;and switch to new data buffer
                 0037
2331    00000846 2C5F               movea.l (sp)+,a6
2332    00000848 60A2               bra.s   RX_stuff_avail          ; And go back and re-compute result
```

```
                        ********************************************************************************
                        *                                                                              *
                        *       routine ctrlblknext:    Routine which determines whether the next      *
                        *               ============    thing to be consumed from the Receive buffer   *
                        *                               is a control block.  THE RESULT OF THIS       *
                        *                               FUNCTION IS NOT VALID UNLESS RX_BUFFER_empty   *
                        *                               RETURNS FALSE!!!                              *
                        *                                                                             *
                        *       At entry:                                                             *
                        *               a2.l =  RX buffer record base address from find_RXBUF         *
                        *               a3.l =  card base address ($00xx0001)                         *
                        *                                                                             *
                        *       Upon exit:                                                            *
                        *               d0.b =  $FF if control block is next, $00 if data is next.    *
                        *               This bashes d2, d5 and a0.                                    *
                        *                                                                             *
                        ********************************************************************************

            0000 084A ctrlblknext equ *
0000084A 6100 F7DE          bsr     gain_access
0000084E 050A 0008          movep.w CTRL_AREA+FILL(a2),d2   ; Check if ctrl buffer is empty
00000852 6100 F832          bsr     release_access
00000856 4280               clr.l   d0
00000858 010A 000C          movep.w CTRL_AREA+EMPTY(a2),d0  ; Fetch ctrl buffer EMPTY pointer
0000085C B440               cmp.w   d0,d2                   ; If equal then return d0.b=$00
0000085E 6728               beq.s   cbn1
00000860 EE58               ror.w   #7,d0
00000862 D08B               add.l   a3,d0
00000864 2040               movea.l d0,a0
00000866 0108 0000          movep.w POINTER(a0),d0          ; Fetch the POINTER field from the

0000086A 4285               clr.l   d5                      ; Setup d5.l=offset
0000086C 1A2C 0037          move.b  which_RXbuf(a4),d5      ; to which Rx buffer
00000870 E985               asl.l   #4,d5                   ; being used

00000872 48E7 8002          movem.l d0/a6,-(sp)
00000876 6100 FE78          bsr     set_RXBUF_a6
0000087A 050E 000C          movep.w EMPTY(a6),d2            ; first ctrl block and compare to the
0000087E 4CDF 4001          movem.l (sp)+,d0/a6

00000882 B440               cmp.w   d0,d2                   ; data buffer EMPTY pointer
00000884 57C0               seq     d0                      ; Then set d0 if equal
00000886 4E75               rts

00000888 4280       cbn1    clr.l   d0
0000088A 4E75               rts
```



```
                        *       *****   ****     ***    *   *    ***    *****   *****   ****
                        *         *     *   *   *   *   **  *   *   *       *       *   *
                        *         *     *   *   *   *   * * *   *   *       *       *   *
                        *         *     ****    *****   *  **    ***    ***     ***     ****
                        *         *     * *     *   *   *   *       *   *       *       *  *
                        *         *     *  *    *   *   *   *   *   *   *       *       *   *
                        *         *     *   *   *   *   *   *    ***    *       *****   *   *


            0000 0006 in_timeout      equ     6  [UNCALIBRATED 1 MS]
            0000 0007 out_timeout     equ     7  [UNCALIBRATED 1 MS]

                        ************************************************
                        *                                              *
                        *   procedure OUTPUT_DATA (                    *
                        *               var SCT:  select_code_table;   *
                        *               PTR:      ^ data_bytes;        *
                        *               COUNT:    longword       );    *
                        *                                              *
                        *   This operation may hang waiting for space. *
                        *                                              *
                        *   This routine calls find_TXBUF and putchars.*
                        *                                              *
                        ************************************************

            0000 088C output_data equ *
0000088C 205F               movea.l (sp)+,a0
0000088E 221F               move.l  (sp)+,d1        ; COUNT
00000890 225F               movea.l (sp)+,a1        ; PTR
00000892 285F               movea.l (sp)+,a4        ; SCT
00000894 4850               pea     (a0)
00000896 266C 0020          movea.l c_adr(a4),a3
0000089A 528B               addq.l  #1,a3

0000089C 2949 003E          move.l  a1,data_address(a4)     ; initialize address/count
000008A0 2941 0042          move.l  d1,data_number(a4)

000008A4 6100 FA96          bsr     check_ov_error          ; Escape if o/v error
000008A8 6100 FD8C          bsr     wait_outxfrdone

000008AC 296C 002E          move.l  timeout(a4),timeout_counter(a4)
             004A
000008B2 297C 0000          move.l  #out_timeout,inner_counter(a4)
             0007 004E

000008BA 6100 F7F4          bsr     find_TXBUF              ; Set up a2.l = buffer descriptor record
                        *                                                       base address
000008BE 6730               beq.s   outdone

000008C0 0838 0001          btst    #timer_present,sysflag2   check for timer      tttt JS 8/11/83
             FEDA
000008C6 672A               beq.s   outtimer                  if got it, use it    tttt JS 8/11/83
```

```
2437
2438    000008C8 4AAC 0042 out_2   tst.l   data_number(a4)          ; And transfer characters until done
2439    000008CC 6722              beq.s   outdone
2440    000008CE 6100 0234         bsr     putchars
2441    000008D2 53AC 004E         subq.l  #1,inner_counter(a4)    ; Test for timeout condition
2442    000008D6 66F0              bne.s   out_2
2443    000008D8 297C 0000         move.l  #out_timeout,inner_counter(a4)
                 0007 004E
2444    000008E0 4AAC 004A         tst.l   timeout_counter(a4)
2445    000008E4 67E2              beq.s   out_2
2446    000008E6 53AC 004A         subq.l  #1,timeout_counter(a4)
2447    000008EA 6700 FA70         beq     time_err                ; if so, escape
2448    000008EE 60D8              bra.s   out_2
2449
2450    000008F0 4E75       outdone rts
2451                        *
2452    000008F2 4AAC 002E outtimer tst.l  timeout(a4)           see if infinit timeout  tttt JS 8/11/83
2453    000008F6 67D0               beq     out_2                if so don't use this    tttt JS 8/11/83
2454    000008F8 1F3C 0001          move.b  #1,-(sp)           else setup timer record   tttt JS 8/11/83
2455    000008FC 2F2C 002E          move.l  timeout(a4),-(sp)                            tttt JS 8/11/83
2456    00000900 4AAC 0042 outtloop tst.l   data_number(a4)    check if all done         tttt JS 8/11/83
2457    00000904 6712               beq.s   outtexit           if so then get out        tttt JS 8/11/83
2458    00000906 6100 01FC          bsr     putchars           else send chars           tttt JS 8/11/83
2459    0000090A 4857               pea     (sp)               push ptr to time rec      tttt JS 8/11/83
2460    0000090C 4EB9 0000          jsr     check_timer        and check the timer       tttt JS 8/11/83
                 0000
2461    00000912 6AEC               bpl     outtloop           if not timeout keep going tttt JS 8/11/83
2462    00000914 6000 FA46          bra     time_err           else do timeout escape    tttt JS 8/11/83
2463    00000918 5C4F       outtexit addq   #6,sp              normal exit -- cleanup    tttt JS 8/11/83
2464    0000091A 4E75               rts                        and return                tttt JS 8/11/83
2465
```



```
2467                        ************************************************
2468                        *                                              *
2469                        *   procedure ENTER_DATA (                     *
2470                        *               var SCT:  select_code_table;   *
2471                        *               PTR:      ^ data_bytes;        *
2472                        *               var COUNT:longword       );    *
2473                        *                                              *
2474                        *  COUNT initially passes the number of bytes  *
2475                        *  which the upper level wants to read.  THE   *
2476                        *  ROUTINE DOES NOT NECESSARILY READ THIS MANY! *
2477                        *  Upon exit COUNT will be reflect the number  *
2478                        *  of data bytes entered, whether or not there *
2479                        *  is an escape.                               *
2480                        *                                              *
2481                        *  escape(F0D): Terminated by reaching a control*
2482                        *     block.  TERM&MODE may be read with STATUS *
2483                        *     9 and 10.                                *
2484                        *                                              *
2485                        *       This routine calls find_RXBUF,         *
2486                        *       getctrlblk, getchars, ctrlblknext, and *
2487                        *       RX_BUFFER_EMPTY.                       *
2488                        *                                              *
2489                        ************************************************
2490
2491             0000 091C enter_data equ  *
2492    0000091C 205F              movea.l (sp)+,a0
2493    0000091E 245F              movea.l (sp)+,a2        ; addr(COUNT)
2494    00000920 225F              movea.l (sp)+,a1        ; PTR
2495    00000922 285F              movea.l (sp)+,a4        ; SCT
2496    00000924 4850              pea     (a0)
2497    00000926 266C 0020         movea.l c_adr(a4),a3
2498    0000092A 528B              addq.l  #1,a3
2499
2500    0000092C 2949 003E         move.l  a1,data_address(a4)     ; initialize address
2501
2502    00000930 6100 FA0A         bsr     check_ov_error          ; Escape if o/v error
2503    00000934 6100 FD0A         bsr     wait_inxfrdone
2504
2505    00000938 296C 002E         move.l  timeout(a4),timeout_counter(a4)
                 004A
2506    0000093E 297C 0000         move.l  #in_timeout,inner_counter(a4)
                 0006 004E
2507
2508    00000946 2952 0042         move.l  (a2),data_number(a4)
2509    0000094A 2F0A              move.l  a2,-(sp)
2510
2511    0000094C 0838 0001         btst    #timer_present,sysflag2   is timer present?  tttt JS 8/11/83
                 FEDA
2512    00000952 6614              bne.s   in_1                        if not, continue    tttt JS 8/11/83
2513    00000954 1F3C 0001         move.b  #1,-(sp)                  else stack time rec    tttt JS 8/11/83
2514    00000958 2F2C 002E         move.l  timeout(a4),-(sp)                                tttt JS 8/11/83
2515
2516    0000095C 600A              bra.s   in_1
2517
2518    0000095E 6100 F7F2 in_0    bsr     eir
2519    00000962 4AAC 0042         tst.l   data_number(a4)         ; See if all characters transferred
2520    00000966 6776              beq.s   in_exit                 ; If so, leave
```

```
2521
2522    00000968 6100 F74A in_1    bsr     find_RXBUF              ; Set up a2.l = buffer descriptor record
2523                       *                                                  base address
2524    0000096C 0838 0001         btst    #timer_present,sysflag2  using timer?          tttt JS 8/11/83
                 FEDA
2525    00000972 6614              bne.s   in_1b                    if not then skip      tttt JS 8/11/83
2526    00000974 4AAC 002E         tst.l   timeout(a4)              infinite timeout?     tttt JS 8/11/83
2527    00000978 6736              beq.s   in_4                     then skip checking    tttt JS 8/11/83
2528    0000097A 4857              pea     (sp)                    push ptr to time re   tttt JS 8/11/83
2529    0000097C 4EB9 0000         jsr     check_timer             and check timer       tttt JS 8/11/83
                 0000
2530    00000982 6A2C              bpl.s   in_4             if not timeout, keep trying   tttt JS 8/11/83
2531    00000984 5C4F              addq    #6,sp            else clean stack              tttt JS 8/11/83
2532    00000986 601A              bra.s   in_1c            and do timeout stuff          tttt JS 8/11/83
2533
2534    00000988 53AC 004E in_1b   subq.l  #1,inner_counter(a4)    ; Test for timeout condition
2535    0000098C 6622              bne.s   in_4
2536    0000098E 297C 0000         move.l  #in_timeout,inner_counter(a4)
                 0006 004E
2537    00000996 4AAC 004A         tst.l   timeout_counter(a4)
2538    0000099A 6714              beq.s   in_4
2539    0000099C 53AC 004A         subq.l  #1,timeout_counter(a4)
2540    000009A0 660E              bne.s   in_4
2541    000009A2 205F       in_1c  movea.l (sp)+,a0
2542    000009A4 2010              move.l  (a0),d0
2543    000009A6 90AC 0042         sub.l   data_number(a4),d0
2544    000009AA 2080              move.l  d0,(a0)
2545    000009AC 6000 F9AE         bra     time_err
2546
2547    000009B0 6100 F7A8 in_4    bsr     dir
2548    000009B4 6100 FE36         bsr     RX_stuff_avail          ; See if buffer is empty
2549    000009B8 4A00              tst.b   d0                      ; If so, just sit here & wait
2550    000009BA 67A2              beq.s   in_0
2551
2552    000009BC 6100 FE8C         bsr     ctrlblknext             ; If a control block is next, then
2553    000009C0 4A00              tst.b   d0
2554    000009C2 6740              beq.s   in_3
2555
2556    000009C4 6100 FD78         bsr     getctrlblk              ; Get it, and check for the special
2557    000009C8 0C2C 00FF         cmpi.b  #255,term(a4)           ; case TERM=255
                 003C
2558    000009CE 6608              bne.s   in_2
2559    000009D0 196C 003D         move.b  mode(a4),which_RXbuf(a4) ;If so, do the buffer switch
                 0037
2560    000009D6 6090              bra.s   in_1                    ; And go back for more
2561
2562    000009D8 396C 003C in_2    move.w  term_and_mode(a4),last_enter_term(a4)
                 0038
2563                       *                                       ; Otherwise save the control block & leave
2564    000009DE 0838 0001 in_exit btst    #timer_present,sysflag2  using timer?          tttt JS 8/11/83
                 FEDA
2565    000009E4 6602              bne.s   in_ex2                   no -- skip ahead      tttt JS 8/11/83
2566    000009E6 5C8F              addq.l  #6,sp                    else clean stack      tttt JS 8/11/83
2567    000009E8 6100 F768 in_ex2  bsr     eir
2568    000009EC 205F              movea.l (sp)+,a0
2569    000009EE 2010              move.l  (a0),d0
2570    000009F0 90AC 0042         sub.l   data_number(a4),d0
```



```
2571    000009F4 2080              move.l  d0,(a0)
2572    000009F6 4AAC 0042         tst.l   data_number(a4)
2573    000009FA 6700 FEF4         beq     outdone                 ; If nonzero then early EOI; escape
2574    000009FE 7016              moveq   #EOD_SEEN,d0
2575    00000A00 6000 F95C         bra     escape
2576
2577    00000A04 4AAC 0042 in_3    tst.l   data_number(a4)         ; see if chars to transfer
2578    00000A08 67D4              beq.s   in_exit                 ; yes, go do it
2579
2580    00000A0A 6100 FD6C         bsr     getchars                ; move some data
2581    00000A0E 6000 FF4E         bra     in_0                    ; & go back to check for ctrl blk
```

```
2583                    ************************************************
2584                    *                                              *
2585                    *   procedure OUTPUT_END (                     *
2586                    *               var SCT:  select_code_table );  *
2587                    *                                              *
2588                    *   Equivalent to the BASIC OUTPUT Sc;END.     *
2589                    *                                              *
2590                    *   This operation may hang waiting for space. *
2591                    *                                              *
2592                    *   This routine calls find_TXBUF and          *
2593                    *   try_sending_EOF.                           *
2594                    *                                              *
2595                    ************************************************
2596
2597           0000 0A12 output_end equ *
2598    00000A12 205F           movea.l (sp)+,a0
2599    00000A14 2B5F           movea.l (sp)+,a4        ; SCT
2600    00000A16 4850           pea     (a0)
2601    00000A18 206C 0020      movea.l c_adr(a4),a3
2602    00000A1C 5348           addq.l  #1,a3
2603
2604    00000A1E 6100 F91C      bsr     check_ov_error          ; Escape if o/v error
2605    00000A22 6100 FC12      bsr     wait_outxfrdone
2606
2607    00000A26 6100 F688      bsr     find_TXBUF              ; Set up a2.1 = buffer descriptor record
2608                            *                                               base address
2609    00000A2A 256C 002E      move.l  timeout(a4),outer_tx_count(a4)
            0046
2610    00000A30 297C 0000      move.l  #sEtimeout,inner_tx_count(a4)
            000B 0052
2611    00000A38 4AAC 0046      tst.l   outer_tx_count(a4)              infinite loop? tttt JS 5/3/84
2612    00000A3C 6708           beq.s   trysend                       yes, normal loop  tttt JS 5/3/84
2613    00000A3E 0838 0001      btst    #timer_present,sysflag2         timer avail?    tttt JS 5/3/84
            FEDA
2614    00000A44 6728           beq.s   try_timer                       if so, use it   tttt JS 5/3/84
2615
2616    00000A46 6100 01F0 trysend bsr  try_sending_EOF
2617    00000A4A 4A00           tst.b   d0
2618    00000A4C 6E1E           bne.s   sentEOF
2619
2620    00000A4E 53AC 0052      subq.l  #1,inner_tx_count(a4)
2621    00000A52 6EF2           bne.s   trysend
2622    00000A54 297C 0000      move.l  #sEtimeout,inner_tx_count(a4)
            000B 0052
2623    00000A5C 4AAC 0046      tst.l   outer_tx_count(a4)
2624    00000A60 67E4           beq.s   trysend
2625    00000A62 53AC 0046      subq.l  #1,outer_tx_count(a4)
2626    00000A66 66DE           bne.s   trysend
2627    00000A68 6000 F8F2      bra     time_err
2628
2629    00000A6C 4E75       sentEOF rts
2630
2631    00000A6E 1F3C 0001 try_timer move.b #1,-(sp)                    setup timer record  tttt JS 5/3/84
2632    00000A72 2F2C 002E      move.l  timeout(a4),-(sp)                                   tttt JS 5/3/84
2633    00000A76 6100 01C0 try_timer2 bsr   try_sending_EOF                                 tttt JS 5/3/84
2634    00000A7A 4A00           tst.b   d0                      successful?                 tttt JS 5/3/84
2635    00000A7C 6616           bne.s   try_timer3              yes, get out                tttt JS 5/3/84
```



```
2636    00000A7E 4857           pea     (sp)                    point to timer rec  tttt JS 5/3/84
2637    00000A80 4EB9 0000      jsr     check_timer             and check time-     tttt JS 5/3/84
            0000
2638    00000A86 6AE6           bpl     try_timer2              if no timeout, loop tttt JS 5/3/84
2639    00000A88 5C4F           addq    #6,sp                   timeout, one more try tttt JS 5/3/84
2640    00000A8A 297C 0000      move.l  #1,outer_tx_count(a4)   with short count    tttt JS 5/3/84
            0001 0046
2641    00000A92 6002           bra     trysend                                     tttt JS 5/3/84
2642    00000A94 5C4F       try_timer3 addq  #6,sp              clean stack         tttt JS 5/3/84
2643    00000A96 4E75           rts                             and return          tttt JS 5/3/84
2644
```

```
2646                    ************************************************
2647                    *                                              *
2648                    *   procedure CONTROL_BFD (                    *
2649                    *               var SCT:  select_code_table;   *
2650                    *               REG:      0..127;              *
2651                    *               VAL:      0..255);             *
2652                    *                                              *
2653                    *   Control register 0 is intercepted and      *
2654                    *   if MODE=0 no action is performed, otherwise *
2655                    *   the card is reset IMMEDIATELY.             *
2656                    *                                              *
2657                    *   This operation may hang waiting for space. *
2658                    *                                              *
2659                    *   The ranges of REG & VAL are not checked    *
2660                    *   for validity.                              *
2661                    *                                              *
2662                    *   This routine calls find_TXBUF and putctrlblk*
2663                    *                                              *
2664                    ************************************************
2665
2666            0000 0A98 control_bfd equ *
2667    00000A98 205F             movea.l (sp)+,a0
2668    00000A9A 321F             move.w  (sp)+,d1        ; VAL
2669    00000A9C 341F             move.w  (sp)+,d2        ; REG
2670    00000A9E 285F             movea.l (sp)+,a4        ; SCT
2671    00000AA0 4850             pea     (a0)
2672    00000AA2 266C 0020        movea.l c_adr(a4),a3
2673    00000AA6 528B             addq.l  #1,a3
2674
2675    00000AA8 6100 F892        bsr     check_ov_error  ; Escape if o/v error
2676                    *                                 (tm) moved 12/02/81
2677    00000AAC 1942 003C        move.b  d2,term(a4)     (tm) ; Intercept CONTROL 0
2678    00000AB0 6710             beq.s   ctrl0           (tm)
2679
2680    00000AB2 6100 FB82        bsr     wait_outxfrdone
2681
2682    00000AB6 1941 003D        move.b  d1,mode(a4)
2683
2684    00000ABA 6100 F5F4        bsr     find_TXBUF              ; Set up a2.l = buffer descriptor record
2685    00000ABE 6000 00B4        bra     putctrlblk              ;               base address
2686
2687    00000AC2 4A01       ctrl0   tst.b   d1
2688    00000AC4 6708             beq.s   ctrldun
2689    00000AC6 6100 FB78        bsr     wait_inxfrdone
2690    00000ACA 6000 F850        bra     do_reset
2691    00000ACE 4E75       ctrldun rts
2692
```



```
2695
2696
2697                    *       *****   *   *   ****    *   *   *****   *****   *****   ****
2698                    *         *     *   *   *   *   *   *   *       *       *       *   *
2699                    *         *      * *    *   *   *   *   *       *       *       *   *
2700                    *         *       *     ****    *   *   *****   *****   *****   ****
2701                    *         *      * *    *   *   *   *   *       *       *       *  *
2702                    *         *     *   *   *   *   *   *   *       *       *       *   *
2703                    *         *     *   *   ****     ***    *       *       *****   *   *
2704
2705
2706
2707            0000 000B sEtimeout       equ     11    [UNCALIBRATED]
2708            0000 000B pcbtimeout      equ     11    [CALIBRATED 1 MS]
2709
2710
2711                    ********************************************************************************
2712                    *                                                                              *
2713                    *       routine TXCTRLBUFFroom: Function which returns the number of byte      *
2714                    *               ==============  positions as yet unused in the Transmit ctrl   *
2715                    *                               Buffer.                                        *
2716                    *                                                                              *
2717                    *       routine TXDATABUFFroom: Function which returns the number of byte      *
2718                    *               ==============  positions as yet unused in the Transmit data   *
2719                    *                               Buffer.                                        *
2720                    *                                                                              *
2721                    *       At entry:                                                              *
2722                    *               a2.l =  TXBUFF base address (shifted, +1+selectcode)           *
2723                    *               a3.l =  card base address ($00xx0001)                          *
2724                    *               d5.l =  TXDATABUFF_SIZE or TXCTRLBUFF_SIZE                     *
2725                    *                       (unshifted, not adjusted)                              *
2726                    *                                                                              *
2727                    *       Upon exit:                                                             *
2728                    *               d0.l =  TXDATABUFF_FILL or TXCTRLBUFF_FILL (unshifted)         *
2729                    *               d3.l =  Number of bytes left                                   *
2730                    *               This also bashes d2.                                           *
2731                    *                                                                              *
2732                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2733                    *                                                                              *
2734                    *       This routine calls gain_access and release_access.                     *
2735                    *                                                                              *
2736                    ********************************************************************************
2737
2738            0000 0AD0 TXCTRLBUFFroom equ *
2739    00000AD0 4280             clr.l   d0                      ; Get garbage out of top of d0&d3
2740    00000AD2 4283             clr.l   d3
2741    00000AD4 010A 0008        movep.w CTRL_AREA+FILL(a2),d0
2742    00000AD8 6100 F550        bsr     gain_access             ; Need access to EMPTY
2743    00000ADC 070A 000C        movep.w CTRL_AREA+EMPTY(a2),d3  ; Fetch pointers (bytes in wrong order)
2744    00000AE0 6010             bra.s   room1
2745
2746            0000 0AE2 TXDATABUFFroom equ *
2747    00000AE2 4280             clr.l   d0                      ; Get garbage out of top of d0&d3
2748    00000AE4 4283             clr.l   d3
2749    00000AE6 010A 0018        movep.w DATA_AREA+FILL(a2),d0
2750    00000AEA 6100 F53E        bsr     gain_access             ; Need access to EMPTY
2751    00000AEE 070A 001C        movep.w DATA_AREA+EMPTY(a2),d3  ; Fetch pointers (bytes in wrong order)
```

```
2752    00000AF2 6100 F592 room1   bsr     release_access
2753
2754    00000AF6 E058              ror.w   #8,d0                   ; Switch bytes in d0 & d3
2755    00000AF8 E05B              ror.w   #8,d3
2756
2757    00000AFA 9640              sub.w   d0,d3                   ; Compute d3 := EMPTY-FILL
2758    00000AFC 5343              subq.w  #1,d3                   ; (EMPTY-FILL-1)
2759    00000AFE 6C02              bge.s   room2
2760    00000B00 DA45              add.w   d5,d3                   ; If negative, add size
2761
2762    00000B02 4E75      room2   rts                             ; Return (EMPTY-FILL-1) mod SIZE
```



```
2764                    ********************************************************************************
2765                    *                                                                              *
2766                    *       routine putchars:       Routine which takes characters from the        *
2767                    *               ========        area sc_subtabletype.data_address sized by     *
2768                    *                               sc_subtabletype.data_number and moves          *
2769                    *                               them to the Transmit buffer.  The number of    *
2770                    *                               characters actually transfered is the minimum  *
2771                    *                               of: (1) the number of characters available;    *
2772                    *                               (2) the number of byte positions left in the   *
2773                    *                               Transmit buffer; and (3) the number of byte    *
2774                    *                               positions in the Transmit buffer until the     *
2775                    *                               wraparound point.  THIS NUMBER CAN BE ZERO.    *
2776                    *                               This alters data_address and data_number to    *
2777                    *                               reflect where to start going next time this    *
2778                    *                               is called.  The entire transfer is done when   *
2779                    *                               data_number goes to zero.                      *
2780                    *                                                                              *
2781                    *       At entry:                                                              *
2782                    *               a2.l =  TX buffer record base address (shifted, +1+selectcode) *
2783                    *               a3.l =  card base address ($00xx0001)                          *
2784                    *               a4.l =  pointer to sc_subtabletype structure                   *
2785                    *                                                                              *
2786                    *       Upon exit:                                                             *
2787                    *               data_number and data_address are updated, plus FILL in the     *
2788                    *               card's Transmit buffer.                                        *
2789                    *               a1, d4 and d5 are left with the values from find_DATA_AREA.    *
2790                    *               This bashes d0, d1, d2, d3, d4, d5, a0, and a1.                *
2791                    *                                                                              *
2792                    *       Interrupts:                                                            *
2793                    *               This does its own enabling/disabling.  Interrupts are left ON. *
2794                    *                                                                              *
2795                    *       This routine uses the card's SEMAPHORE to gain access.                 *
2796                    *                                                                              *
2797                    *       This routine calls gain_access, release_access, TXDATABUFFroom,        *
2798                    *       and find_DATA_AREA.                                                    *
2799                    *                                                                              *
2800                    ********************************************************************************
2801
2802            0000 0B04 putchars equ     *
2803    00000B04 6100 F5E6          bsr     find_DATA_AREA          ; Setup a1 = data buffer base addr
2804                    *                                                   d4 = end of data buffer addr
2805                    *                                                   d5 = TXDATABUFF_SIZE
2806    00000B08 6100 F650          bsr     dir
2807    00000B0C 61D4               bsr     TXDATABUFFroom          ; d3.l = available buffer positions
2808    00000B0E 2200               move.l  d0,d1                   ; d0.l = d1.l = TXDATABUFF_FILL
2809    00000B10 2004               move.l  d4,d0
2810    00000B12 0280 0000          andi.l  #$0000FFFE,d0
                FFFE
2811    00000B18 E280               asr.l   #1,d0                   ; d0.l = unshifted TXDATABUFF_END
2812    00000B1A 9081               sub.l   d1,d0                   ; d0.l = remaining positions to wrap
2813
2814    00000B1C B680               cmp.l   d0,d3                   ; If d0>d3 then set d0 := d3
2815    00000B1E 6E02               bgt.s   pc1
2816    00000B20 2003               move.l  d3,d0
2817
2818    00000B22 2428 0042 pc1      move.l  data_number(a4),d2      ; Fetch number of chars avail into d2
2819    00000B26 B480               cmp.l   d0,d2                   ; If d0>d2 then set d0 := d2
```

```
2820    00000B28 6E02              bgt.s   pc2
2821    00000B2A 2002              move.l  d2,d0
2822
2823    00000B2C 2600      pc2     move.l  d0,d3                   ; d3.l saves number of chars actually
2824                       *                                       ; transferred below
2825    00000B2E 6740              beq.s   pcdone                  ; If zero, no work to be done
2826    00000B30 5340              subq.w  #1,d0                   ; Make offset correct for dbf instr.
2827    00000B32 206C 003E         movea.l data_address(a4),a0     ; Get character pointer into a0
2828
2829    00000B36 E349              lsl.w   #1,d1
2830    00000B38 D28E              add.l   a3,d1
2831    00000B3A 48E7 0040         movem.l a1,-(sp)                ; Save a1 so we can use the register
2832    00000B3E 2241              movea.l d1,a1                   ; Now a1 is useable pointer
2833
2834    00000B40 1298      pcloop  move.b  (a0)+,(a1)              ; Transfer a character & bump source ptr
2835    00000B42 5449              addq.w  #2,a1                   ; Bump destination pointer (odd bytes)
2836    00000B44 51C8 FFFA         dbf     d0,pcloop               ; Then decrement d0 & loop
2837
2838    00000B48 9483              sub.l   d3,d2
2839    00000B4A 2942 0042         move.l  d2,data_number(a4)      ; Now store adjusted number and
2840    00000B4E 2948 003E         move.l  a0,data_address(a4)     ; address fields
2841
2842    00000B52 2209              move.l  a1,d1                   ; Move 68000 FILL pointer into d1
2843    00000B54 4CDF 0200         movem.l (sp)+,a1                ; Restore a1 before we forget!
2844    00000B58 B881              cmp.l   d1,d4                   ; Now check to see if FILL was moved
2845    00000B5A 6602              bne.s   pc3                     ; past end of buffer.  If so, set to
2846    00000B5C 2209              move.l  a1,d1                   ; the front of the buffer.
2847    00000B5E 0881 0000 pc3     bclr    #0,d1                   ; Fix up the 68000 pointer to be the
2848    00000B62 EF59              rol.w   #7,d1                   ; card's type of pointer
2849    00000B64 610C F4C4         bsr     gain_access
2850    00000B68 038A 0018         movep.w d1,DATA_AREA+FILL(a2)   ; Remember d1 = card's FILL pointer.
2851    00000B6C 6100 F518         bsr     release_access
2852    00000B70 6000 F5E0 pcdone  bra     eir
```



```
2854                       ********************************************************************************
2855                       *                                                                              *
2856                       *       routine putctrlblk:     Routine which puts a control block into the    *
2857                       *                 ==========    Transmit buffer area of the card.  The         *
2858                       *                               appropriate pointers are updated to reflect    *
2859                       *                               the control block.  This routine also contains *
2860                       *                               a timeout mechanism which will be adjusted     *
2861                       *                               to the proper values later.  If a timeout      *
2862                       *                               occurs, an escape is done with NO DAMAGE to    *
2863                       *                               the buffer.  The only thing that can cause the *
2864                       *                               timeout is < 4 positions left in the control   *
2865                       *                               buffer.  SEMAPHORE timeout is not handled      *
2866                       *                               by this routine.                               *
2867                       *                                                                              *
2868                       *       At entry:                                                              *
2869                       *               sc_subtabletype.term =  TERM field for control block (8 bits)  *
2870                       *               sc_subtabletype.mode =  MODE field for control block (8 bits)  *
2871                       *               a2.l =  TX buffer record base address (shifted, +1+selectcode) *
2872                       *               a3.l =  card base address ($00xx0001)                          *
2873                       *               a4.l =  pointer to sc_subtabletype structure                   *
2874                       *                                                                              *
2875                       *       Upon exit:                                                             *
2876                       *               FILL in the card's transmit control buffer is updated.         *
2877                       *               a1, d4 and d5 are left with the values from find_CTRL_AREA.    *
2878                       *               This bashes d0, d1, d2, d3, d4, d5, a0 and a1.                 *
2879                       *               This uses inner/outer_tx_count for computing timeouts.         *
2880                       *                                                                              *
2881                       *       Interrupts:                                                            *
2882                       *               This does its own enabling/disabling.  Interrupts are left ON. *
2883                       *                                                                              *
2884                       *       This routine uses the card's SEMAPHORE to gain access.                 *
2885                       *                                                                              *
2886                       *       This routine calls TXCTRLBUFFroom, escape, gain_access, find_CTRL_AREA,*
2887                       *       eir, dir and release_access.                                           *
2888                       *                                                                              *
2889                       ********************************************************************************
2890
2891             0000 0B74 putctrlblk equ  *
2892    00000B74 6100 F57E         bsr     find_CTRL_AREA          ; Setup a1 = ctrl buffer base addr
2893                       *                                                 d4 = end of ctrl buffer addr
2894                       *                                                 d5 = TXCTRLBUFF_SIZE
2895    00000B78 296C 002E         move.l  timeout(a4),outer_tx_count(a4)
                  0046
2896    00000B7E 297C 0000         move.l  #pcbtimeout,inner_tx_count(a4) ; Load timeout value
                  000B 0052
2897                       *
2898    00000B88 0838 0001         btst    #timer_present,sysflag2      timer present?    tttt JS 8/11/83
                  FEDA
2899    00000B8C 676C              beq.s   pcbtime                   if so then use it    tttt JS 8/11/83
2900                       *
2901    00000B8E 6100 F5CA pcbwait bsr     dir
2902    00000B92 6100 FF3C         bsr     TXCTRLBUFFroom          ; Get d3 = #bytes available in buffer
2903    00000B96 0C83 0000         cmpi.l  #4,d3                   ; and d0 = CTRLBUFF_FILL (unshifted)
                  0004
2904    00000B9C 6C22              bge.s   roomok                  ; If >=4 bytes, can go ahead!
2905    00000B9E 6100 F5B2         bsr     eir
2906    00000BA2 53AC 0052         subq.l  #1,inner_tx_count(a4)
```

```
00000BA6 66E6           bne.s   pcbwait                 ; Loop, then if it times out give an
00000BA8 297C 0000      move.l  #pcbtimeout,inner_tx_count(a4)
         000B 0052
00000BB0 4AAC 0046      tst.l   outer_tx_count(a4)
00000BB4 67D8           beq.s   pcbwait
00000BB6 53AC 0046      subq.l  #1,outer_tx_count(a4)
00000BBA 66D2           bne.s   pcbwait
00000BBC 6000 F79E      bra     time_err                ; escape(timeout).

00000BC0 E348       roomok  lsl.w   #1,d0               ; Make CTRLBUFF_FILL into a 68000
00000BC2 D08B           add.l   a3,d0                   ; pointer
00000BC4 2040           movea.l d0,a0                   ; Put in a0 to use it.

00000BC6 010A 0018      movep.w DATA_AREA+FILL(a2),d0   ; Get the DATA_FILL pointer to put
00000BCA 0188 0000      movep.w d0,POINTER(a0)          ; into the POINTER FIELD
00000BCE 302C 003C      move.w  term_and_mode(a4),d0
00000BD2 0188 0004      movep.w d0,TERMFIELD(a0)
00000BD6 D1FC 0000      adda.l  #CTRLBLKSIZE,a0         ; Bump pointer by TWO bytes
         0008

00000BDC 2208           move.l  a0,d1                   ; Move 68000 FILL pointer into d1
00000BDE B881           cmp.l   d1,d4                   ; Now check to see if FILL was moved
00000BE0 6602           bne.s   pcb1                    ; past end of buffer.  If so, set to
00000BE2 2209           move.l  a1,d1                   ; the front of the buffer.
00000BE4 0381 0000 0000 pcb1    bclr    #0,d1           ; Fix up the 68000 pointer to be the
00000BE8 EF59           rol.w   #7,d1                   ; card's type of pointer
00000BEA 6100 F43E      bsr     gain_access
00000BEE 038A 0008      movep.w d1,CTRL_AREA+FILL(a2)
00000BF2 6100 F492      bsr     release_access
00000BF6 6000 F55A      bra     eir
                *
00000BFA 4AAC 002E pcbtime tst.l    timeout(a4)         see if infinite timeout     tttt JS 8/11/83
00000BFE 678E           beq     pcbwait                 if so use other loops       tttt JS 8/11/83
00000C00 1F3C 0001      move.b  #1,-(sp)                else setup time record      tttt JS 8/11/83
00000C04 2F2C 002E      move.l  timeout(a4),-(sp)                                   tttt JS 8/11/83
00000C08 6100 F550 pcbtloop bsr     dir                 loop checks copied from     tttt JS 8/11/83
00000C0C 6100 FEC2      bsr     TXCTRLBUFFroom          code above                  tttt JS 8/11/83
00000C10 0C83 0000      cmpi.l  #4,d3                                               tttt JS 8/11/83
         0004
00000C16 6C1C           bge.s   troomok                 ok -- leave                 tttt JS 8/11/83
00000C18 6100 F538      bsr     eir                                                 tttt JS 8/11/83
00000C1C 4857           pea     (sp)                    push ptr to time rec        tttt JS 8/11/83
00000C1E 4EB9 0000      jsr     check_timer             timeout?                    tttt JS 8/11/83
         0000
00000C24 6AE2           bpl     pcbtloop                no, do loop again           tttt JS 8/11/83
00000C26 5C4F           addq    #6,sp                   yes, clean stack and do     tttt JS 5/3/84
00000C28 297C 0000      move.l  #1,outer_tx_count(a4)   quick final check           tttt JS 5/3/84
         0001 0046
00000C30 6000 FF5C      bra     pcbwait                                             tttt JS 5/3/84
00000C34 5C4F       troomok addq    #6,sp               normal exit -- cleanup stk  tttt JS 8/11/83
00000C36 6088           bra     roomok                  and return                  tttt JS 8/11/83
```



```
                        ************************************************
                        *                                              *
                        *   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 0C38 try_sending_EOF equ *
00000C38 7204           moveq   #4,d1
00000C3A 4280           clr.l   d0
00000C3C 102A FFFC      move.b  TXENDBLOCKSPACE-TXBUFF(a2),d0
00000C40 671C           beq.s   sE3                     ; If it's zero jump down & wait for 4
00000C42 2200           move.l  d0,d1                   ; bytes in the control queue
00000C44 6100 F4A6 sE1      bsr     find_DATA_AREA      ; Setup a1 = data buffer base addr
                *                                                   d4 = end of data buffer addr
                *                                                   d5 = TXDATABUFF_SIZE
00000C48 6100 F510 sE1loop bsr      dir
00000C4C 6100 FE94          bsr     TXDATABUFFroom      ; Now hang until enough space becomes
00000C50 B681               cmp.l   d1,d3               ; available in the data queue
00000C52 6C08               bge.s   sE2
00000C54 6100 F4FC          bsr     eir
00000C58 4280           noroom  clr.l   d0
00000C5A 4E75               rts

00000C5C 7208           sE2     moveq   #8,d1           ; if TXENDBLOCKSPACE#0 then wait for
                *                                       ; 8 bytes, not 4.
00000C5E 6100 F494 sE3      bsr     find_CTRL_AREA      ; Setup a1 = ctrl buffer base addr
                *                                                   d4 = end of ctrl buffer addr
                *                                                   d5 = TXCTRLBUFF_SIZE
00000C62 6100 F4F6 sE3loop bsr      dir
00000C66 6100 FE68          bsr     TXCTRLBUFFroom      ; Now hang until enough space becomes
00000C6A B681               cmp.l   d1,d3               ; available in the ctrl queue
00000C6C 6C06               bge.s   sE4
00000C6E 6100 F4E2          bsr     eir
00000C72 60E4               bra.s   noroom

00000C74 4A2A FFFC sE4      tst.b   TXENDBLOCKSPACE-TXBUFF(a2)  ; There's enough room now!! If zero
00000C78 6726               beq.s   sE6                 ; then just send 1 block below
00000C7A 397C 0501          move.w  #$0501,term_and_mode(a4)
         003C
00000C80 6100 FEF2          bsr     putctrlblk
00000C84 42AC 0042          clr.l   data_number(a4)         ; Followed by some space
00000C88 196A FFFC          move.b  TXENDBLOCKSPACE-TXBUFF(a2),data_number+3(a4)
         0045
```

```
 3008    00000C8E 297C FFFF          move.l  #$FFFF0000,data_address(a4) ; kluge so it isn't left pointing
                  0000 003E
 3009                    *                                               to nowhere & get bus error!
 3010    00000C96 6100 FE6C sE5      bsr     putchars
 3011    00000C9A 4AAC 0042          tst.l   data_number(a4)             ; Hang until all sent
 3012    00000C9E 66F6               bne.s   sE5
 3013    00000CA0 397C 0500 sE6      move.w  #$0500,term_and_mode(a4)
                  003C
 3014    00000CA6 6100 FECC          bsr     putctrlblk
 3015    00000CAA 7001               moveq   #1,d0
 3016    00000CAC 4E75               rts
 3017
 3018
 3019                                end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

Description

DGL_AUTL provides assembly language utility routines for DGL scaled moves and draws.

```
PASS 1 COMPLETE. ERRORS: 0
    1                       *
    2                       * Pascal work station graphic library
    3                       *   scaling routine
    4                       *
    5                       * Module       = DGL_AUTL
    6                       * Programer    = BJS
    7                       * Date         = 8/27/82
    8                       *
    9                       * Purpose : To provide low level asmb routines for DGL
   10                       *
   11                       *
   12                       *  Rev history
   13                       *
   14                       *    Created  -  8-27-82  - BJS
   15                       *    Modified - 11-23-82  - BJS  Removed ck for static link on proc calls
   16                       *                2-10-82  - BJS  Removed gcb, and proc var parms for performance
   17                       *                6-27-83  - BJS  Removed module init body, now in
   18                       *                                module DGL_IBODY
   19                       *
   20                       *
   21                       *     (c) Copyright Hewlett-Packard Company, 1983.
   22                       *     All rights are reserved.  Copying or other
   23                       *     reproduction of this program except for archival
   24                       *     purposes is prohibited without the prior
   25                       *     written consent of Hewlett-Packard Company.
   26                       *
   27                       *
   28                       *                 RESTRICTED RIGHTS LEGEND
   29                       *
   30                       *     Use, duplication, or disclosure by the Government
   31                       *     is subject to restrictions as set forth in
   32                       *     paragraph (b) (3) (B) of the Rights in Technical
   33                       *     Data and Computer Software clause in
   34                       *     DAR 7-104.9(a).
   35                       *
   36                       *     HEWLETT-PACKARD COMPANY
   37                       *     Fort Collins, Colorado
   38                       *
   39                       *
   40                         MNAME DGL_AUTL
   41
   42                         src module dgl_autl;
   43                         src export
   44                         src PROCEDURE dgl_scaled_move;
   45                         src PROCEDURE dgl_scaled_draw;
   46                         src end;
   47
   48                       *  Define entry points
   49
   50  00000000               rorg 0
   51                         def     dgl_autl_dgl_autl
   52                         def     dgl_autl_dgl_scaled_move
   53                         def     dgl_autl_dgl_scaled_draw
   54
   55                       *  Define externals
   56
   57                         refa    dgl_vars
   58                         lmode   dgl_vars
```



```
   59
   60
   61                       *  Define ASMB control information
   62
   63                         nosyms
   64
   65                       *  Define constants
   66
   67            0000 0005 global  equ a5
   68            0000 0003 gle_gcb equ a3
   69                       *
   70                       **      global variables
   71                       *
   72            FFFF FF60 scale_factors equ dgl_vars-160
   73            FFFF FF5E short_defaults equ dgl_vars-162
   74            FFFF FFFF system_init equ dgl_vars-1
   75            FFFF FFFE disp_init   equ dgl_vars-2
   76            FFFF FFFD loc_init    equ dgl_vars-3
   77            FFFF FF46 gle_gcb_def equ dgl_vars-186
   78
   79                         include ASM_TYPES
   80            0000 0018 AWAIT_BLANKING equ 24
   81            0000 018E BACKGROUND equ 398
   82            0000 0020 BUFFER_MODE equ 32
   83            0000 029C CALC_SOFT_TEXT_XFORM equ 668
   84            0000 029A CALC_TEXT_XFORM equ 666
   85            0000 01D2 CHAR_HEIGHT equ 466
   86            0000 01E6 CHAR_JUST_X equ 486
   87            0000 01EA CHAR_JUST_Y equ 490
   88            0000 0028 CHAR_SIZE equ 40
   89            0000 01B0 CHAR_SIZES equ 432
   90            0000 01D6 CHAR_SPACE equ 470
   91            0000 01CE CHAR_WIDTH equ 462
   92            0000 0030 CLEAR equ 48
   93            0000 0038 CLIP_LIMITS equ 56
   94            0000 01F2 CLIP_LIMITS_XMAX equ 498
   95            0000 01EE CLIP_LIMITS_XMIN equ 494
   96            0000 01FA CLIP_LIMITS_YMAX equ 506
   97            0000 01F6 CLIP_LIMITS_YMIN equ 502
   98            0000 0196 COLOR_MAP_SUPPORT equ 406
   99            0000 0190 COMPLEMENT_SUPPORT equ 400
  100            0000 01AA CONT_LINESTYLES equ 426
  101            0000 023A COSX_TABLE equ 570
  102            0000 024A COSY_TABLE equ 586
  103            0000 0208 CURRENT_BUFFER_MODE equ 520
  104            0000 0212 CURRENT_COLOR_INDEX equ 530
  105            0000 01FE CURRENT_CURSOR_STATE equ 510
  106            0000 0200 CURRENT_CURSOR_X equ 512
  107            0000 0204 CURRENT_CURSOR_Y equ 516
  108            0000 0216 CURRENT_DRAWING_MODE equ 534
  109            0000 0214 CURRENT_FILL_INDEX equ 532
  110            0000 020A CURRENT_LINESTYLE equ 522
  111            0000 0210 CURRENT_LINESTYLE_MODE equ 528
  112            0000 020C CURRENT_LINESTYLE_PATTERN equ 524
  113            0000 0218 CURRENT_LINEWIDTH equ 536
  114            0000 020E CURRENT_PATTERN_LENGTH equ 526
  115            0000 0220 CURRENT_POLYGON_BLUE equ 544
```

```
0000 021A CURRENT_POLYGON_COLOR equ 538
0000 021E CURRENT_POLYGON_GREEN equ 542
0000 021C CURRENT_POLYGON_RED equ 540
0000 01B2 CURREN_POS_X equ 434
0000 01B6 CURREN_POS_Y equ 438
0000 0040 CURSOR equ 64
0000 0048 DEFINE_COLOR_MAP equ 72
0000 0050 DEFINE_DRAWING_MODE equ 80
0000 014C DEVICE_BUF equ 332
0000 0154 DEVICE_INFO equ 340
0000 0158 DEVICE_INFO_CHAR_COUNT equ 344
0000 0150 DEV_DEP_STUFF equ 336
0000 016C DISPLAY_HANDLER_CHAR_COUNT equ 364
0000 0166 DISPLAY_HANDLER_NAME equ 358
0000 0186 DISPLAY_MAX_X equ 390
0000 018A DISPLAY_MAX_Y equ 394
0000 017E DISPLAY_MIN_X equ 382
0000 0182 DISPLAY_MIN_Y equ 386
0000 015E DISPLAY_NAME equ 350
0000 0164 DISPLAY_NAME_CHAR_COUNT equ 356
0000 016E DISPLAY_RES_X equ 366
0000 0176 DISPLAY_RES_Y equ 374
0000 01A0 DITHER_SUPPORT equ 416
0000 0058 DRAW equ 88
0000 0118 DUMMY_XXX equ 280
0000 01BA END_X equ 442
0000 01BE END_Y equ 446
0000 0194 ERASE_SUPPORT equ 404
0000 015A ERROR_RETURN equ 346
0000 0060 FILL_INDEX_COLOR equ 96
0000 0068 FLUSH_BUFFER equ 104
0000 01A6 GAMUT equ 422
0000 0070 GET_COLOR_MAP equ 112
0000 0080 GET_POLYGON_INFO equ 128
0000 0078 GET_RASTER equ 120
0000 0088 GLOAD equ 136
0000 0090 GRAPHICS_ON_OFF equ 144
0000 0098 GSTORE equ 152
0000 00A0 INDEX_COLOR equ 160
0000 0000 INF01 equ 0
0000 0004 INF02 equ 4
0000 0008 INF03 equ 8
0000 000C INF04 equ 12
0000 0010 INFO_PTR1 equ 16
0000 0014 INFO_PTR2 equ 20
0000 00A8 INQ_P1P2 equ 168
0000 0148 IOCB equ 328
0000 0120 IO_INQ_TIMEOUT equ 288
0000 0128 IO_READ equ 296
0000 0130 IO_SET_TIMEOUT equ 304
0000 0138 IO_TERM equ 312
0000 0140 IO_WRITE equ 320
0000 01CC KATA equ 460
0000 00B8 LINESTYLE equ 184
0000 00B0 LINEWIDTH equ 176
0000 01AE LINEWIDTHS equ 430
0000 01DA LINE_SPACE equ 474
```



```
0000 00C0 MARKER equ 192
0000 01C8 MARKER_HEIGHT equ 456
0000 00C8 MARKER_SIZE equ 200
0000 01C2 MARKER_TYPE equ 450
0000 01C4 MARKER_WIDTH equ 452
0000 00D0 MOVE equ 208
0000 0192 NON_DOMINANT_SUPPORT equ 402
0000 0222 OLD_A5 equ 546
0000 0226 OLD_A6 equ 550
0000 00D8 OUTPUT_ESCAPE1 equ 216
0000 00E0 OUTPUT_ESCAPE0 equ 224
0000 01A2 PALLETTE equ 418
0000 00E8 POLYGON equ 232
0000 019C POLYGON_FILL_FACTOR equ 412
0000 019E POLYGON_SOLID_FILL equ 414
0000 0198 POLYGON_SUPPORT equ 408
0000 019A REDEF_BACKGROUND equ 410
0000 00F0 SET_MARKER equ 240
0000 026A SINX_TABLE equ 618
0000 027A SINY_TABLE equ 634
0000 02D2 SOFT_CLIP_CPX equ 722
0000 02D6 SOFT_CLIP_CPY equ 726
0000 02C0 SOFT_CLIP_SAVEX0 equ 704
0000 02C4 SOFT_CLIP_SAVEX1 equ 708
0000 02C8 SOFT_CLIP_SAVEY0 equ 712
0000 02CC SOFT_CLIP_SAVEY1 equ 716
0000 02D0 SOFT_CLIP_SWITCH equ 720
0000 02A4 SOFT_FONT_PTR equ 676
0000 02A8 SOFT_TEXT_TEMP1 equ 680
0000 02AC SOFT_TEXT_TEMP2 equ 684
0000 015C SPOOLING equ 348
0000 00F8 TEXT equ 248
0000 01E2 TEXT_COS_DIR equ 482
0000 0100 TEXT_DIR equ 256
0000 0108 TEXT_JUST equ 264
0000 0232 TEXT_LINE_X equ 562
0000 0236 TEXT_LINE_Y equ 566
0000 01DE TEXT_SIN_DIR equ 478
0000 022A TEXT_SPACE_X equ 554
0000 022E TEXT_SPACE_Y equ 558
0000 0110 TEXT_SPACING equ 272
0000 02B8 UNCLIPPED_DRAW equ 696
0000 02B0 UNCLIPPED_MOVE equ 688
0000 01AC VECT_LINESTYLES equ 428


                    ********************************************************************************
                    *
                    *  integer scale
                    *
                    *  a0 -> display_offset d2
                    *        window_delta   d1
                    *        display_delta  d0
                    *
                    *  end_x := (end_x * display_delta) / window_delta + display_offset;
                    *  end_y := . . .
```

```
PAGE   5 [3.0] 12/26/84 22:38:04  ASSEMBLY OF DGL_AUTL.TEXT                                  *** File name: DGL_AUTL ***

  230                       *
  231               0000 0000 dgl_autl_dgl_scaled_draw equ *
  232
  233    0000000C 236D FF46         movea.l  gle_gcb_def(global),gle_gcb   get gcb
  234
  235    00000004 4A2D FF5E         tst.b    short_defaults(global)    is scaling neeced?
  236    00000008 662C              bne.s    done1
  237
  238    0000000A 202B 01BA         move.l   end_x(gle_gcb),d6
  239
  240    0000000E 41ED FF60         lea      scale_factors(global),a0
  241    00000012 4C98 003F         movem.w  (a0)+,d0-d5   get scaling factors
  242
  243    00000016 CDC0              muls     d0,d6
  244    00000018 8DC1              divs     d1,d6
  245    0000001A 6964              bvs.s    tdb
  246    0000001C DD42              add.w    d2,d6
  247
  248    0000001E 4BC6              ext.l    d6
  249    00000020 2746 01BA         move.l   d6,end_x(gle_gcb)
  250
  251    00000024 202B 01BE         move.l   end_y(gle_gcb),d6
  252
  253    00000028 CDC3              muls     d3,d6
  254    0000002A 8DC4              divs     d4,d6
  255    0000002C 6952              bvs.s    tdb
  256    0000002E DD45              add.w    d5,d6
  257
  258    00000030 4BC6              ext.l    d6
  259    00000032 2746 01BE         move.l   d6,end_y(gle_gcb)
  260
  261               0000 0036 done1  equ *
  262    00000036 2F0B              move.l   gle_gcb,-(sp)     setup to pass gcb ptr
  263    00000038 206B 0058         movea.l  draw(gle_gcb),a0  no static links
  264    0000003C 4E90              jsr      (a0)              call draw
  265
  266    0000003E 4E75              rts
  267
  268               0000 0040 dgl_autl_dgl_scaled_move equ *
  269
  270    00000040 236D FF46         movea.l  gle_gcb_def(global),gle_gcb   get gcb
  271
  272    00000044 4A2D FF5E         tst.b    short_defaults(global)    is scaling needed?
  273    00000048 662C              bne.s    done2
  274
  275    0000004A 202B 01BA         move.l   end_x(gle_gcb),d6
  276
  277    0000004E 41ED FF60         lea      scale_factors(global),a0
  278    00000052 4C98 003F         movem.w  (a0)+,d0-d5   get scaling factors
  279
  280    00000056 CDC0              muls     d0,d6
  281    00000058 8DC1              divs     d1,d6
  282    0000005A 6924              bvs.s    tdb
  283    0000005C DD42              add.w    d2,d6
  284
  285    0000005E 4BC6              ext.l    d6
  286    00000060 2746 01BA         move.l   d6,end_x(gle_gcb)
```

```
PAGE   6 [3.0] 12/26/84 22:38:04  ASSEMBLY OF DGL_AUTL.TEXT                                  *** File name: DGL_AUTL ***

  287
  288    00000064 202B 01BE         move.l   end_y(gle_gcb),d6
  289
  290    00000068 CDC3              muls     d3,d6
  291    0000006A 8DC4              divs     d4,d6
  292    0000006C 6912              bvs.s    tdb
  293    0000006E DD45              add.w    d5,d6
  294
  295    00000070 4BC6              ext.l    d6
  296    00000072 2746 01BE         move.l   d6,end_y(gle_gcb)
  297
  298               0000 0076 done2  equ *
  299    00000076 2F0B              move.l   gle_gcb,-(sp)     setup to pass gcb ptr
  300    00000078 206B 00D0         movea.l  move(gle_gcb),a0  no static links
  301    0000007C 4E90              jsr      (a0)              call draw
  302
  303    0000007E 4E75              rts
  304
  305    00000080 4E76      tdb     trapv                      overflow
  306    00000082 4E75              rts
  307
  308                       ********************************************************************************
  309                       *
  310               0000 0084 dgl_autl_dgl_autl equ *
  311
  312    00000084 4E75              rts
  313
  314                        end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

Description

DGL_IBODY contains a routine to set DGL globals to a known, "uninitialized" state.

```
PAGE   1 [3.0] 12/20/84 22:39:07  ASSEMBLY OF DGL_IBODY.TEXT                                   *** File name: DGL_IBODY ***

PASS 1 COMPLETE. ERRORS: 0
    1                           *
    2                           * Pascal work station graphic library initialization body
    3                           *
    4                           * Module       = DGL_IBODY
    5                           * Programer    = BJS
    6                           * Date         = 8/27/82
    7                           *
    8                           * Purpose : To set globals state to known value (uninitialized)
    9                           *
   10                           *  Rev history
   11                           *
   12                           *    Created  -  6-27-83  - BJS
   13                           *    Modified -
   14                           *
   15                           *
   16                           *     (c) Copyright Hewlett-Packard Company, 1983.
   17                           *     All rights are reserved.  Copying or other
   18                           *     reproduction of this program except for archival
   19                           *     purposes is prohibited without the prior
   20                           *     written consent of Hewlett-Packard Company.
   21                           *
   22                           *
   23                           *                    RESTRICTED RIGHTS LEGEND
   24                           *
   25                           *     Use, duplication, or disclosure by the Government
   26                           *     is subject to restrictions as set forth in
   27                           *     paragraph (b) (3) (B) of the Rights in Technical
   28                           *     Data and Computer Software clause in
   29                           *     DAR 7-104.9(a).
   30                           *
   31                           *     HEWLETT-PACKARD COMPANY
   32                           *     Fort Collins, Colorado
   33                           *
   34                           *  This module is called during module initialization ('pre-run") time.
   35                           *  It sets the state variables for the graphics library to indicate that
   36                           *  the library is not initialized.  This is done to indicated to the
   37                           *  library that pointers in global space may no longer be valid (i.e.
   38                           *  dynamic space used by the library no longer belongs to the library
   39                           *  after program termination).  This procedure resets variables, which
   40                           *  might not be reset by the loading process since the library can be
   41                           *  'P loaded'.
   42                           *
   43                           *  When using segments, this module may be dummied out to allow graphics
   44                           *  routines to be called in many segments without re-initializing the
   45                           *  library in each segment.  However, the library and the display
   46                           *  (graphics_init, display_init) MUST be initialized in the main
   47                           *  program.  This is so the dynamic memory for the library will remain
   48                           *  between segment calls.
   49                           *
   50                                  MNAME DGL_IBODY
   51
   52                                  src module dgl_IBODY;
   53                                  src export
   54                                  src end;
   55
   56                           *  Define entry points
   57
   58    00000000                      rorg 0
```

```
PAGE   2 [3.0] 12/20/84 22:39:07  ASSEMBLY OF DGL_IBODY.TEXT                                   *** File name: DGL_IBODY ***

   59                                 def     dgl_ibody_dgl_ibody
   60
   61                           *  Define externals
   62
   63                                  refa    dgl_vars
   64                                  lmode   dgl_vars
   65
   66
   67                           *  Define ASMB control information
   68
   69                                  nosyms
   70
   71                           *  Define constants
   72
   73              0000 0005 global  equ a5
   74              0000 0003 gle_gcb equ a3
   75                           *
   76                           **      global variables
   77                           *
   78              FFFF FFFF system_init equ dgl_vars-1
   79              FFFF FFFE disp_init   equ dgl_vars-2
   80              FFFF FFFD loc_init    equ dgl_vars-3
   81
   82                           ********************************************************************************
   83                           *
   84              0000 0000 dgl_ibody_dgl_ibody equ *
   85
   86                           *  Initialize system variables to not enabled.
   87
   88    00000000 422D FFFF        clr.b system_init(global)
   89    00000004 422D FFFE        clr.b disp_init(global)
   90    00000008 422D FFFD        clr.b loc_init(global)
   91    0000000C 4E75             rts
   92
   93                                 end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

DISCINT

Description

DISCINT contains assembly language low-level drivers.

Usage

DISCINT is used for the 98625 interface.

Requirements

DI_DRV and COMASM.

```
PASS 1 COMPLETE. ERRORS: 0
```



```
                        ********************************************************************************
                        *
                        *       copyright (c) 1984 by HEWLETT-PACKARD COMPANY
                        *
                        ********************************************************************************
                        *
                        *
                        *       iolib     extdi
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *
                        *       library - iolib
                        *       author  -
                        *       phone   -
                        *
                        *       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  - 08/10/83
                        *       release - 5/15/84
                        *
                        *
                        *       source  - IOLIB:DISCINT.TEXT
                        *       object  - IOLIB:DISCINT.CODE
                        *
                        *
                        ********************************************************************************
                        *
                        *
                        *       released
                        *       version         3.0
                        *
                        *
                        ********************************************************************************
```

```
43                      ********************************************************************************
44                      *                                                                              *
45                      *                                                                              *
46                      *       bug fix history         - after release                                *
47                      *                                                                              *
48                      *                                                                              *
49                      *       bug #   by  / on        loc             description                    *
50                      *       -----   ----------      --------------  ---------------------          *
51                      *       tttt    ---------       di_wfc           Changes to use timer          *
52                      *               8/10/83         di_IFC           on UMM boards if avail.       *
53                      *                               input_tfr_term                                 *
54                      *                                                                              *
55                      ********************************************************************************
```



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

```
PAGE   5 [3.0] 12/20/84 21:30:57 IOLIB EXTDI - 98625 DISC INTERFACE DRIVERS                *** File name: DISCINT ***
  103                     ********************************************************************************
  104                     *
  105                     *       symbols for export as procedure names
  106                     *
  107                     ********************************************************************************
  108
  109                                     def     extdi_extdi
  110                                     def     extdi_edi_init
  111                                     def     extdi_edi_isr
  112                                     def     extdi_edi_rdb
  113                                     def     extdi_edi_wtb
  114                                     def     extdi_edi_rdw
  115                                     def     extdi_edi_wtw
  116                                     def     extdi_edi_rds
  117                                     def     extdi_edi_wtc
  118                                     def     extdi_edi_tfr
  119
  120                                     def     extdi_edi_send
  121                                     def     extdi_edi_ppoll
  122                                     def     extdi_edi_set
  123                                     def     extdi_edi_clr
  124                                     def     extdi_edi_test
  125                                     def     extdi_edi_end
  126
  127
  128
  129                     ********************************************************************************
  130                     *
  131                     *       symbols for import - common assembly language routines
  132                     *
  133                     *       the routines are in the module common_assembly and powerup
  134                     *
  135                     ********************************************************************************
  136
  137                                     refa    dropdma         give up DMA resource
  138                                     refa    getdma          actually get DMA
  139                                     refa    testdma         check to see if DMA is available
  140                                     refa    logint          branch to user isr
  141                                     refa    logeot          branch to user eot
  142                                     refa    stbsy           set buffer busy
  143                                     refa    stclr           set buffer not busy
  144                                     refa    DMA_stbsy       set buffer DMA busy
  145                                     refa    itxfr           is there a tfr active ?
  146                                     refa    abort_io        kill any tfr active
  147                                     refa    wait_tfr        timed wait for tfr active
  148                                     refa    check_tfr       timed wait for tfr - direction
  149                                     refa    check_timer     use for timeouts with hw timer tttt jws
  150                                     refa    delay_timer     use for all delays          tttt jws
  151
  152
  153                                     lmode   dropdma,getdma,testdma,logint,logeot,stbsy
  154                                     lmode   stclr,DMA_stbsy,itxfr,abort_io,wait_tfr,check_tfr
  155                                     lmode   check_timer,delay_timer                     tttt jws
  156
  157

PAGE   6 [3.0] 12/20/84 21:30:57 IOLIB EXTDI - COMMON EQUATES AND DEFINITIONS              *** File name: DISCINT ***
  160                             include COMDCL
```

```
163                     ********************************************************************************
164                     *
165                     *       modified:  02/22/82  JPC   added parm to user EOT & ISR proc's
166                     *                  08/01/83  JS    added timer_present and sysflag2 equ's
167                     *
168                     ********************************************************************************
169                     *
170                     *       HPL CONVENTIONS
171                     *
172                     *
173                     *         Much of this code is taken intact from the 9826 HPL
174                     *         language system EIO ROM ( extended I/O ROM ).
175                     *
176                     *         The Pascal that will be calling this code uses
177                     *         the stack for parameter passage.  The HPL code
178                     *         uses the Ax and Dx registers for all parameters.
179                     *         The Pascal driver entry points on the previous pages
180                     *         take care of getting the parameters into the correct
181                     *         registers.
182                     *
183                     *
184                     *       GENERAL HPL ENTRY/EXIT CONDITIONS:
185                     *
186                     *         A1.L = CARD ADDRESS
187                     *         A2.L = DRIVER TEMP ADDRESS
188                     *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
189                     *
190                     *
191                     *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
192                     *
193                     *         A3.L = BUFFER CONTROL BLOCK ADDRESS
194                     *                In addition to the A1/A2 convention, Pascal will use
195                     *                A3 for a pointer to the buffer control block.
196                     *                The HPL system kept much of the transfer
197                     *                information in the s.c. temps.
198                     *
199                     *         TIMEOUT(A2) = contains timeout information
200                     *                Timeout was a global temp in HPL and a timeout
201                     *                generated an error.
202                     *                In PASCAL each card has a timeout value stored in
203                     *                its temporary area.  A timeout error
204                     *                generates an ESCAPE ( which can be trapped ).
205                     *
206                     *
207                     ********************************************************************************
```



```
209                     ********************************************************************************
210                     *
211                     *
212                     *         DRIVER TEMPS TEMPLATE
213                     *
214                     *           OFFSET FROM A2
215                     *
216                     *           HPL DECLARATIONS ( MODIFIED )
217                     *
218                     *
219                     ********************************************************************************
220           0000 0000 ISR_ENTRY   EQU 0     ..19      PASCAL ISR LINK & UNLINK area
221           0000 0014 USER_ISR    EQU 20              user ISR:  do NOT change the proc/stat link/parm ordering!!!
222           0000 0014 H_ISR_PR    EQU 20    ..23            procedure ptr
223           0000 0018 H_ISR_SL    EQU 24    ..27            static link
224           0000 001C H_ISR_PM    EQU 28    ..31            parameter
225           0000 0020 C_ADR       EQU 32    ..35      card address
226           0000 0024 BUF1_OFF    EQU 36    ..39      buffer pointer offset
227           0000 0028 BUF0_OFF    EQU 40    ..43      buffer pointer offset
228           0000 002C EIRB_OFF    EQU 44              eir byte
229           0000 002D IO_SC       EQU 45              select code ( i.e. 7, 22, etc. )
230           0000 002E TIMEOUT     EQU 46    ..49      timeout value
231                     *                                   =0 : no timeout
232                     *                                   #0 : value of timeout
233           0000 0032 MA_W        EQU 50    ..51      word access to my address
234           0000 0033 MA          EQU 51              byte access to my address
235           0000 0034 AVAIL_OFF   EQU 52    ..??      standard space taken from temps
236                     *               52    ..83      normal cards ( 32  bytes )
237                     *               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

0000 0001 TIMER_PRESENT  EQU   1        JS  8/1/83  SYSFLAG2_BIT -- 0=>TIMER PRESENT
FFFF FEDA SYSFLAG2       EQU   $FFFFFEDA          JS  8/1/83
```

```
                        *
                        * 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 0402                      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 1680                       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 01EE                  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 0238                  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 04C4                  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 0342                  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 025E                  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 0280                  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 02D6                  bra     di_clr               clear the line
```

```
PAGE  15 [3.0] 12/26/84 21:30:57 IOLIB EXTDI - PASCAL ENTRY POINTS                    *** File name: DISCINT ***

                        *
                        * 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 F2F8                      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
```

```
PAGE  16 [3.0] 12/26/84 21:30:57 IOLIB EXTDI - DISC INTERFACE CARD DECLARATIONS      *** File name: DISCINT ***

                        ********************************************************************************
                        *
                        *       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 6100                           bsr.s   di_init_s               start software reset
00000122 6600 0270                      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                      move.b  #1,fifo(a1)             enable a 1 byte counted transfer
         0015
00000194 611E           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 0008                      btst    #8,status(a1)           tagged with EOI?
         0017
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                      move    #ioe_error,esc_code(a5) save system esc code
         FFFE
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    di,intmask(a1)          unmask the appropriate interrupt bit
0000020E 41E9 0011                      lea     intreg(a1),a0           point to the interrupt register
                        *
                        * quick low-overhead loop
                        *
00000212 243C 0000                      move.l  #254,d2                 quick loop counter
         00FE
                        *
                        * Count changed from 127 to 254                  tttt jws 8/10/83
                        *
00000218 0310           di_wfc_quick    btst    d1,(a0)                 condition met?
0000021A 662E                           bne.s   di_wfc_done             branch if so
0000021C 51CA FFFA                      dbra    d2,di_wfc_quick         loop until quick count expires
                        *
                        * timed wait loop
                        *
00000220 242A C02E                      move.l  timeout(a2),d2          current timeout
00000224 6720                           beq.s   di_wfc_infinite         branch if infinite timeout
00000226 0838 C001                      btst    #timer_present,sysflag2 check if timer     tttt jws
         FEDA
0000022C 6728                           beq.s   di_wfc_timer            br if got it       tttt jws
0000022E C4FC 01EB                      mulu    #491,d2                 loop iterations per millisecond

00000232 0310           di_wfc_timed    btst    d1,(a0)                 condition met?
00000234 6614                           bne.s   di_wfc_done             branch if so
00000236 5382                           subq.l  #1,d2                   decrement the loop counter
00000238 62F8                           bhi     di_wfc_timed            loop until count expired

0000023A 03A9 0013                      bclr    d1,intmask(a1)          re-mask the appropriate interrupt bit
0000023E 137C 0080                      move.b  #$80,cardcontrol(a1)    re-enable card interrupts
         0003
00000244 60A4                           bra     di_tmo                  escape with timeout error
                        *
                        * infinite wait loop
                        *
00000246 0310           di_wfc_infinite btst    d1,(a0)                 condition met?
00000248 67FC                           beq     di_wfc_infinite         loop until so
                        *
                        * wait for condition done
                        *
0000024A 03A9 0013      di_wfc_done     bclr    d1,intmask(a1)          re-mask the appropriate interrupt bit
```

```
  798    0000024E 137C 0080                move.b  #$80,cardcontrol(a1)    re-enable card interrupts
              0003
  799    00000254 4E75                     rts                             done!
  800                      *
  801                      *  Wait using the timer                                        tttt jws
  802                      *
  803    00000256 1F3C 0001 di_wfc_timer    move.b  #1,-(sp)        setup timer record  tttt jws 8/10/83
  804    0000025A 2F02                     move.l  d2,-(sp)                            tttt jws 8/10/83
  805    0000025C 0310      di_wfc_tloop   btst    d1,(a0)         check condition     tttt jws 8/10/83
  806    0000025E 6610                     bne.s   di_wfc_texit    if true, exit       tttt jws 8/10/83
  807    00000260 4857                     pea     (sp)            else call timer     tttt jws 8/10/83
  808    00000262 4EB9 0000                jsr     check_timer     check routine       tttt jws 8/10/83
              0000
  809    00000268 6AF2                     bpl     di_wfc_tloop    ok -- loop          tttt jws 8/10/83
  810    0000026A 5C4F                     addq    #6,sp           timeout, clean stk  tttt jws 5/2/84
  811    0000026C 740A                     moveq   #10,d2          try again with a    tttt jws 5/2/84
  812    0000026E 60C2                     bra     di_wfc_timed    short count         tttt jws 5/2/84
  813    00000270 5C4F      di_wfc_texit   addq    #6,sp           cleanup stack       tttt jws 8/10/83
  814    00000272 6006                     bra     di_wfc_done     and continue        tttt jws 8/10/83
```



```
  816                      ********************************************************************************
  817                      *
  818                      *       di_rds
  819                      *
  820                      *         read status
  821                      *
  822                      *         PASCAL routine
  823                      *
  824                      ********************************************************************************
  825
  826    00000274 B27C 0008 di_rds          cmp     #8,d1                   register within range?
  827    00000278 621C                      bhi.s   rds_err                 branch if not
  828    0000027A D241                      add     d1,d1                   two bytes per table entry
  829    0000027C 323B 1006                 move    rds_table(d1),d1        load routine offset
  830    00000280 4EFB 1002                 jmp     rds_table(d1)           jump to the appropriate routine
  831
  832                      *
  833                      * rds jump table
  834                      *
  835    00000284 0018      rds_table       dc      rds_0-rds_table         status 0
  836    00000286 0012                      dc      rds_1-rds_table         status 1
  837    00000288 0012                      dc      rds_2-rds_table         status 2
  838    0000028A 001C                      dc      rds_3-rds_table         status 3
  839    0000028C 0012                      dc      rds_4-rds_table         status 4
  840    0000028E 0012                      dc      rds_5-rds_table         status 5
  841    00000290 0030                      dc      rds_6-rds_table         status 6
  842    00000292 0012                      dc      rds_7-rds_table         status 7
  843    00000294 0012                      dc      rds_8-rds_table         status 8
  844
  845              0000 0296 rds_1           equ *
  846              0000 0296 rds_2           equ *
  847              0000 0296 rds_4           equ *
  848              0000 0296 rds_5           equ *
  849              0000 0296 rds_7           equ *
  850              0000 0296 rds_8           equ *
  851
  852    00000296 7013      rds_err         moveq   #bad_rds,d0             bad read status
  853    00000298 6000 FF52                 bra     esc_err
  854
```

```
856                         *
857                         * status 0 - card ID
858                         *
859  0000029C 7008          rds_0           moveq   #8,d0                   Simon's ID
860  0000029E 4E75                          rts
861
862
863                         *
864                         * status 3  -  controller status & address
865                         *
866  000002A0 7000          rds_3           moveq   #0,d0                   ZZZZZZZZ ZZZZZZZZ
867  000002A2 1029 0017                     move.b  status(a1),d0           ZZZZZZZZ 76543210
868  000002A6 E918                          rol.b   #4,d0                   ZZZZZZZZ 32107654
869  000002A8 E348                          lsl     #1,d0                   ZZZZZZZ3 2107654Z
870  000002AA ED08                          lsl.b   #6,d0                   ZZZZZZZ3 4ZZZZZZZ
871  000002AC E248                          lsr     #1,d0                   ZZZZZZZZ 34ZZZZZZ
872  000002AE D02A 0033                     add.b   ma(a2),d0               ZZZZZZZZ 34ZAAAAA
873  000002B2 4E75                          rts
874
875
876                         *
877                         * status 6  -  interface status
878                         *
879  000002B4 61EA          rds_6           bsr     rds_3           ZZZZZZZZ ZZZZZZZZ ZZZZZZZZ LLLLLLLL
880  000002B6 E098                          ror.l   #8,d0           LLLLLLLL ZZZZZZZZ ZZZZZZZZ ZZZZZZZZ
881  000002B8 1029 0017                     move.b  status(a1),d0   LLLLLLLL ZZZZZZZZ ZZZZZZZZ 76543210
882  000002BC E358                          rol     #1,d0           LLLLLLLL ZZZZZZZZ ZZZZZZZ7 6543210Z
883  000002BE E298                          ror.l   #1,d0           ZLLLLLLL LZZZZZZZ ZZZZZZZZ 76543210
884  000002C0 E418                          ror.b   #2,d0           ZLLLLLLL LZZZZZZZ ZZZZZZZZ 10765432
885  000002C2 E298                          ror.l   #1,d0           2ZLLLLLL LLZZZZZZ ZZZZZZZZ Z1076543
886  000002C4 E518                          rol.b   #2,d0           2ZLLLLLL LLZZZZZZ ZZZZZZZZ 076543Z1
887  000002C6 E298                          ror.l   #1,d0           12ZLLLLL LLLZZZZZ ZZZZZZZZ Z076543Z
888  000002C8 E958                          rol     #4,d0           12ZLLLLL LLLZZZZZ ZZZZZ076 543ZZZZZ
889  000002CA E898                          ror.l   #4,d0           ZZZZ12ZL LLLLLLLZ ZZZZZZZZ Z076543Z
890  000002CC E618                          ror.b   #3,d0           ZZZZ12ZL LLLLLLLZ ZZZZZZZZ 43ZZ0765
891  000002CE E298                          ror.l   #1,d0           5ZZZZ12Z LLLLLLLL ZZZZZZZZ Z43ZZ076
892  000002D0 4840                          swap    d0              ZZZZZZZZ Z43ZZ076 5ZZZZ12Z LLLLLLLL
893  000002D2 4E75                          rts                             whew!!!
894
```



```
896                         ********************************************************************************
897                         *
898                         *       di_wtc
899                         *
900                         *         write control
901                         *
902                         *         entry:  d0.w = parameter
903                         *
904                         ********************************************************************************
905
906  000002D4 0C41 0001 di_wtc          cmpi    #1,d1                   register within range?
907  000002D8 62BC                      bhi.s   rds_err                 branch if not
908  000002DA D241                      add     d1,d1                   two bytes per table entry
909  000002DC 4EFB 1002                 jmp     hwtctbl(d1)             jump to the appropriate routine
910
911
912  000002E0 6002          hwtctbl     bra.s   di_wtc_rst              control 0 - do a reset
913  000002E2 6016                      bra.s   di_rqs                  control 1 - set SRQ response
914                         *          bra.s   di_wtc_ppc              control 2 - ppoll configure
915                         *          bra.s   di_wtc_sma              control 3 - set my addr
916                         *          bra.s   rds_err                 control 4 - not used
917                         *          bra.s   rds_err                 control 5 - enable interrupts
918
919
920                         *
921                         * software reset
922                         *
923  000002E4 102A 0033 di_wtc_rst      move.b  ma(a2),d0               else use previous address
924
925  000002E8 1F29 0017 hpl_wtc1        move.b  status(a1),-(sp)        save controller active state.
926  000002EC 6100 FE3A                 bsr     di_init_s               do software reset
927  000002F0 081F 0004                 btst    #4,(sp)+                were we active controller?
928  000002F4 6702                      beq.s   hpl_wtc2                if not, skip
929  000002F6 4E71                      nop     ????????????????????????????????????????????????
930  000002F8 4E75          hpl_wtc2    rts                             else done
931
932
933                         *
934                         * request service
935                         *
936  000002FA 08AA C006 di_rqs          bclr    #6,flags(a2)            assume rsv = 0 in this new byte
             0039
937  00000300 1200                      move.b  d0,d1                   if rsv bit in the new byte is indeed
938  00000302 0881 0006                 bclr    #6,d1                   zero, then just output the new byte.
939  00000306 6706                      beq.s   di_rqs2
940                         *          move.b  d1,di_spoll(a1)         else first write the byte with rsv
941  00000308 08EA 0006                 bset    #6,flags(a2)            clear. remember that rsv is set.
             0039
942  0000030E 1280          di_rqs2     move.b  d0,(a1) <<<di_spoll(a1)         write the byte with rsv correct.
943  00000310 322A 003A                 move    ppollmsk(a2),d1         go update the parallel poll response
944  00000314 4E4B          hpl_wtc4    trap    #11                     get into supervisor mode   scs
945                         * scs       move    sr,-(sp)                disable isr's while we figure
946  00000316 007C 2700                 ori     #$2700,sr               out which mask to set based on
947  0000031A 082A 0006                 btst    #6,flags(a2)            current rsv bit.
             0039
948  00000320 6702                      beq.s   hpl_wtc5                if rsv = 0, use right byte
949  00000322 E059                      ror     #8,d1                   else use left byte
```

```
 950    00000324 1281       hpl_wtc$        move.b  d1,(a1) <<<di_ppoll(a1)
 951    00000326 4E5F                       move    (sp)+,sr                restore user mode       scs
 952    00000328 4E75                       rts                                                     scs
 953                        * scs           rte                             re-enable isr's and return
 954
```



```
 956                        ********************************************************************************
 957                        *
 958                        *       di_p_poll
 959                        *
 960                        *         conduct parallel poll
 961                        *
 962                        *         if not active controller give error
 963                        *         else value returned in d0.b
 964                        *
 965                        *         hpl routine
 966                        *
 967                        ********************************************************************************
 968
 969    0000032A 0829 0004 di_p_poll       btst    #4,status(a1)           active controller?
                  0017
 970    00000330 6700 FE9C                 beq     di_notactl              branch if not
 971
 972    00000334 6100 FECA                 bsr     di_wait_fi              wait for fifo idle
 973
 974    00000338 4229 0003                 clr.b   cardcontrol(a1)         disable card interrupts
 975    0000033C 08E9 0005                 bset    #5,intmask(a1)          unmask the poll response bit for ABI's sake
                  0013
 976    00000342 4229 0017                 clr.b   status(a1)              high-order bits
 977    00000346 50E9 001D                 st      ppolmask(a1)            look at all response bits
 978    0000034A 4229 001F                 clr.b   ppolsense(a1)          normal sense for all
 979    0000034E 1029 0015                 move.b  fifo(a1),d0             get the response
 980    00000352 08A9 0005                 bclr    #5,intmask(a1)          re-mask the poll response bit
                  0013
 981    00000358 137C 0080                 move.b  #$80,cardcontrol(a1)    re-enable card interrupts
                  0003
 982
 983    0000035E 4E75                      rts
 984
```

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

00000360 B27C 0007 di_set          cmp     #7,d1                   register within range?
00000364 6200 FE64                 bhi     di_sc_err               branch if not
00000368 D241                      add     d1,d1                   two bytes per table entry
0000036A 323B 1006                 move    set_table(d1),d1        routine offset
0000036E 4EFB 1002                 jmp     set_table(d1)           jump to the appropriate routine

00000372 0010      set_table       dc      di_REN-set_table        REN  - set REN
00000374 0022                      dc      di_IFC-set_table        IFC  - pulse IFC (set REN/clr ATN)
00000376 FE58                      dc      di_sc_err-set_table     SRQ  - error
00000378 004E                      dc      di_EOI-set_table        EOI  - pulse EOI on next byte out
0000037A FE58                      dc      di_sc_err-set_table     NRFD - error
0000037C FE58                      dc      di_sc_err-set_table     NDAC - error
0000037E FE58                      dc      di_sc_err-set_table     DAV  - error
00000380 FE58                      dc      di_sc_err-set_table     ATN  - error


                        *
                        * set REN
                        *
00000382 0829 0003 di_REN          btst    #3,status(a1)           system controller?
         0017
00000388 6700 FE48                 beq     di_notsctl              branch if not
0000038C 08E9 0005                 bset    #5,control(a1)          set REN
         0019
00000392 4E75                      rts


                        *
                        * assert IFC for 100us
                        *
00000394 0829 0003 di_IFC          btst    #3,status(a1)           system controller?
         0017
0000039A 6700 FE36                 beq     di_notsctl              branch if not

0000039E 41E9 0019                 lea     control(a1),a0          point to the control register
000003A2 0890 0005                 bclr    #5,(a0)                 clear REN
000003A6 08D0 0004                 bset    #4,(a0)                 assert IFC
                        *  tttt jws     moveq   #80,c0              must hold IFC at least 100us...
                        *  tttt jws     dbra    d0,*                to satisfy the bus standard
000003AA 2F3C 0000                 move.l  #100,-(sp)              use the timer       tttt jws
         0064
000003B0 4EB9 0000                 jsr     delay_timer             routine for this    tttt jws 8/10/83
         0000
000003B6 0890 0004                 bclr    #4,(a0)                 release IFC
000003BA 08D0 0005                 bset    #5,(a0)                 set REN

000003BE 4E75                      rts
```



```


                        *
                        * assert EOI on the next wtb
                        *
000003C0 50EA 0036 di_EOI          st      EOI_out(a2)             set the flag
000003C4 4E75                      rts
```

```
1047                        ********************************************************************************
1048                        *
1049                        *       di_clr
1050                        *
1051                        *         set an hpib line false
1052                        *
1053                        *         PASCAL routine
1054                        *
1055                        ********************************************************************************
1056
1057    000003C6 B.7C 0007 di_clr          cmp     #7,d1                   register within range?
1058    000003CA 6200 FDFE                 bhi     di_sc_err               branch if not
1059    000003CE D.41                      add     d1,d1                   two bytes per table entry
1060    000003D0 323B 1006                 move    clear_table(d1),d1      routine offset
1061    000003D4 4EFB 1002                 jmp     clear_table(d1)         jump to the appropriate routine
1062
1063    000003D8 0010      clear_table     dc      di_local-clear_table    REN  - clear REN
1064    000003DA 0020                      dc      di_dmyrts-clear_table   IFC  - nothing
1065    000003DC FDF2                      dc      di_sc_err-clear_table   SRQ  - error
1066    000003DE 0020                      dc      di_dmyrts-clear_table   EOI  - nothing
1067    000003E0 FDF2                      dc      di_sc_err-clear_table   NRFD - error
1068    000003E2 FDF2                      dc      di_sc_err-clear_table   NDAC - error
1069    000003E4 FDF2                      dc      di_sc_err-clear_table   DAV  - error
1070    000003E6 FDF2                      dc      di_sc_err-clear_table   ATN  - error
1071
1072
1073                        *
1074                        * clear REN
1075                        *
1076    000003E8 0829 0003 di_local        btst    #3,status(a1)           system controller?
             0017
1077    000003EE 6700 FDE2                 beq     di_notsctl              branch if not
1078    000003F2 08A9 0005                 bclr    #5,control(a1)          clear REN
             0019
1079    000003F8 4E75      di_dmyrts       rts
1080
```



```
1082                        ********************************************************************************
1083                        *
1084                        *       di_test
1085                        *
1086                        *         get an hpib line's state
1087                        *
1088                        *         entry :       d1 = line parameter
1089                        *                             0 = REN
1090                        *                             1 = IFC
1091                        *                             2 = SRQ
1092                        *                             3 = EOI
1093                        *                             4 = NRFD
1094                        *                             5 = NDAC
1095                        *                             6 = DAV
1096                        *                             7 = ATN
1097                        *
1098                        *         PASCAL routine
1099                        *
1100                        ********************************************************************************
1101
1102    000003FA 6000 FDCE di_test         bra             di_sc_err
1103
```

```
1105                        ********************************************************************************
1106                        *
1107                        *       di_r6out
1108                        *
1109                        *         emulation of r6 out for hp-ib
1110                        *
1111                        *         entry:  d0 = byte to output
1112                        *
1113                        *         exit:   if not active controller, sts cleared and error bit set
1114                        *                 else operation is done and any addressing decoded.
1115                        *
1116                        *         hpl routine ( modified )
1117                        *
1118                        ********************************************************************************
1119
1120    000003FE 4EB9 0000 di_r6out        jsr     wait_tfr                if a tfr is active wait till it isn't
                  0000
1121
1122    00000404 0829 0004                 btst    #4,status(a1)           active controller?
                  0017
1123    0000040A 6700 FDC2                 beq     di_notact1              branch if not
1124
1125    0000040E 6100 FDF0                 bsr     di_wait_fi              wait for fifo idle
1126    00000412 137C 0040                 move.b  #$40,status(a1)         following byte is a command
                  0017
1127    00000418 1340 0015                 move.b  d0,fifo(a1)             send it
1128
1129    0000041C 4E75                      rts
1130
```



```
1133                        ********************************************************************************
1134                        *
1135                        *       di_isr
1136                        *
1137                        *         interrupt service routine for hp-ib cards
1138                        *
1139                        *         entry : a1,a2 are set up
1140                        *
1141                        *         the isr will track down the buffer control block
1142                        *
1143                        *         hpl routine ( modified )
1144                        *
1145                        ********************************************************************************
1146
1147    0000041E 08AA 0001 di_isr          bclr    #1,flags(a2)            clear user isr pending flag
                  0039
1148    00000424 6100 0100                 bsr     di_isr1                 process the interrupting condition(s)
1149    00000428 4EB9 0000                 jsr     itxfr                   go see if transfer is active
                  0000
1150    0000042E 6716                      beq.s   isr_end                 if not, then we are done.
1151    00000430 B23C 0003                 cmp.b   #tt_burst,d1            if frw is active, go process it
1152    00000434 6700 00E2                 beq     di_frw
1153    00000438 B23C 0001                 cmp.b   #tt_int,d1              if int then go process it
1154    0000043C 6700 00C4                 beq     di_buf
1155    00000440 B23C 0002                 cmp.b   #tt_DMA,d1              if DMA
1156    00000444 6710                      beq.s   di_isrdma
1157    00000446 08AA 0001 isr_end         bclr    #1,flags(a2)            if isr pending then do it
                  0039
1158    0000044C 6706                      beq.s   di_isr_ex
1159    0000044E 4EB9 0000                 jsr     logint
                  0000
1160    00000454 4E75      di_isr_ex       rts                             otherwise return
1161
```

```
1163                            *
1164                            * DMA transfer interrupt:
1165                            *
1166    00000456 2D6A 0040 di_isrdma        movea.l DMA_arm_addr(a2),a4     DMA arm & status register address
1167    0000045A 4A2B 000D                  tst.b   tdir_off(a3)            test direction
1168    0000045E 6B32                       bne.s   isrdma_out              branch if output
1169
1170
1171                            *
1172                            * input DMA transfer
1173                            *
1174    00000460 41EB 0020                  lea     tfil_off(a3),a0         point to the buffer fill pointer
1175
1176    00000464 0829 0006                  btst    #6,status(a1)           last byte in tagged with EOI?
                 0017
1177    0000046A 56EA 0035                  sne     EOI_in(a2)              remember it
1178
1179    0000046E 082C 0000                  btst    #0,1(a4)                DMA channel unarmed?
                 0001
1180    00000474 670A                       beq.s   isrdma_unarmed          branch if so
1181
1182    00000476 0829 0001                  btst    #1,intreg(a1)           fifo idle?
                 0011
1183    0000047C 6608                       bne.s   isrdma_in_term          branch of so (terminate tfr)
1184    0000047E 60C6                       bra     isr_end                 otherwise, keep going
1185
1186    00000480 4AAA 003C isrdma_unarmed   tst.l   DMA_count(a2)           more to transfer?
1187    00000484 6F38                       bgt.s   isrdma_restart          branch if so
1188
1189    00000486 6100 021C isrdma_in_term   bsr     input_tfr_term          clean up the fifo's if possible
1190    0000048A 08A9 0001                  bclr    #1,intmask(a1)          re-mask fifo idle condition
                 0013
1191    00000490 6046                       bra.s   isrdma_term
1192
1193
1194                            *
1195                            * output DMA transfer
1196                            *
1197    00000492 41EB 001C isrdma_out       lea     temp_off(a3),a0         point to the buffer empty pointer
1198
1199    00000496 082C 0000                  btst    #0,1(a4)                DMA channel still armed?
                 0001
1200    0000049C 66A8                       bne.s   isr_end                 branch if so; keep going
1201
1202    0000049E 08E9 0007                  bset    #7,control(a1)          re-set 8-bit mode if necessary
                 0019
1203    000004A4 4229 0017                  clr.b   status(a1)              clear the hi-order bits
1204
1205    000004A8 0C6A 0001                  cmpi.w  #1,DMA_count(a2)        [ cmpi.l #$10000,DMA_count(a2) ]
                 003C
1206    000004AE 6D28                       blt.s   isrdma_term             branch if no more to transfer
1207
1208    000004B0 6E0C                       bgt.s   isrdma_restart          branch if additional bursts remain
1209    000004B2 4A2B 000B                  tst.b   tend_off(a3)            last burst: EOI tag set?
1210    000004B6 6706                       beq.s   isrdma_restart          branch if not, otherwise...
1211    000004B8 08A9 0007                  bclr    #7,control(a1)          clear 8-bit mode (tag last byte with EOI)
                 0019
```


1212

```
PAGE  35 [3.0] 12/26/84 21:30:57 IOLIB EXTDI - DISC INTERFACE INTERRUPT SERVICE ROUTINE      *** File name: DISCINT ***

 1214                          *
 1215                          * DMA transfer restart
 1216                          *
 1217    000004BE 536A 003C isrdma_restart  subq.w  #1,DMA_count(a2)        [ sub.l #$10000,DMA_count(a2) ]
 1218    000004C2 202B 0010                 move.l  tcnt_off(a3),d0        previous burst length
 1219    000004C6 D190                      add.l   d0,(a0)                update the appropriate buffer pointer
 1220    000004C8 277C 0001                 move.l  #$10000,tcnt_off(a3)   next burst length
                  0000 0010
 1221    000004D0 38AA 0044                 move    DMA_arm_word(a2),(a4)  re-arm the DMA channel
 1222    000004D4 6000 FF70                 bra     isr_end                done!
 1223
 1224
 1225                          *
 1226                          * DMA transfer termination
 1227                          *
 1228    000004D8 137C 0080 isrdma_term     move.b  #$80,cardcontrol(a1)   disable card DMA
                  0003
 1229    000004DE 2F08                      move.l  a0,-(sp)               save a0
 1230    000004E0 4EB9 0000                 jsr     dropdma                free the DMA channel
                  0000
 1231    000004E6 205F                      movea.l (sp)+,a0               restore a0
 1232    000004E8 262B 0010                 move.l  tcnt_off(a3),d3        intended number of bytes transfered
 1233    000004EC 9684                      sub.l   d4,d3                  actual number of bytes transferred
 1234    000004EE D790                      add.l   d3,(a0)                update the appropriate buffer pointer
 1235    000004F0 D8AA 003C                 add.l   DMA_count(a2),d4       total remaining count
 1236    000004F4 2744 0010                 move.l  d4,tcnt_off(a3)        record it
 1237    000004F8 4EB9 0000                 jsr     stclr                  mark the tfr done and log branch
                  0000
 1238    000004FE 6000 FF46                 bra     isr_end                done!
 1239
 1240
 1241
 1242
 1243                          *
 1244                          * interrupt transfer processing:
 1245                          *
 1246    00000502 4A2B 000D di_buf          tst.b   tdir_off(a3)           which direction transfer?
 1247    00000506 6602                      bne.s   di_bufo                 branch if output
 1248
 1249
 1250                          *
 1251                          * buffered input:
 1252                          *
 1253    00000508 4E71         di_bufi      nop
 1254
 1255                          *
 1256                          * buffered output:
 1257                          *
 1258              0000 050A di_bufo        equ *
 1259
 1260                          *
 1261                          * int and frw transfer termination
 1262                          *
 1263              0000 050A di_ti_term     equ *
 1264
 1265    0000050A 103C 0000 di_to_term      move.b  #0,d0                  disable other interrupts
 1266                          *                    bsr     di_eir
```

```
PAGE  36 [3.0] 12/26/84 21:30:57 IOLIB EXTDI - DISC INTERFACE INTERRUPT SERVICE ROUTINE      *** File name: DISCINT ***

 1267    0000050E 4EB9 0000                 jsr     stclr                  mark the buffer finished
                  0000
 1268    00000514 6000 FF30                 bra     isr_end               end of isr
 1269
 1270
 1271                          *
 1272                          * fast r/w transfer processing:
 1273                          *
 1274    00000518 007C 2700 di_frw          ori     #$2700,sr                disable all other ints
 1275                          *                                        the PASCAL system will re-enable&rte
 1276    0000051C 4A2B 000D                 tst.b   tdir_off(a3)           which direction transfer?
 1277    00000520 6602                      bne.s   di_frwo                 skip if output
 1278
 1279                          *
 1280                          * fast r/w input:
 1281                          *
 1282    00000522 4E71         di_fri       nop
 1283
 1284                          *
 1285                          * fast r/w output:
 1286                          *
 1287              0000 0524 di_frwo        equ *
 1288    00000524 60E4                      bra     di_to_term             else we are done!
```

```
1290                        ********************************************************************************
1291                        *
1292                        *       isr1
1293                        *
1294                        *         the following routine does all the grunt work for the isr. it is
1295                        *         separated out so it can be called from background.
1296                        *
1297                        ********************************************************************************
1298
1299    00000526 41E9 0011  di_isr1         lea     intreg(a1),a0           point to the interrupt condition register
1300    0000052A 49E9 0013                  lea     intmask(a1),a4          point to the interrupt mask register
1301                        *
1302                        * DCL
1303                        *
1304    0000052E 0810 0000                  btst    #0,(a0)                 DCL interrupt?
1305    00000532 6708                       beq.s   di_no_DCL               branch if not
1306    00000534 6100 0032                  bsr     di_log                  else do normal logging
1307    00000538 0894 0000                  bclr    #0,(a4)                 mask off the interrupt
1308             0000 053C  di_no_DCL       equ *
1309                        *
1310                        * SRQ
1311                        *
1312    0000053C 0810 0004                  btst    #4,(a0)                 SRQ interrupt?
1313    00000540 6708                       beq.s   di_no_SRQ               branch if not
1314    00000542 6100 0024                  bsr     di_log                  else do normal logging
1315    00000546 0894 0004                  bclr    #4,(a4)                 mask off the interrupt
1316             0000 054A  di_no_SRQ       equ *
1317                        *
1318                        * POL
1319                        *
1320    0000054A 0810 0005                  btst    #5,(a0)                 POL interrupt?
1321    0000054E 6708                       beq.s   di_no_POL               branch if not
1322    00000550 6100 0016                  bsr     di_log                  else do normal logging
1323    00000554 0894 0005                  bclr    #5,(a4)                 mask off the interrupt
1324             0000 0558  di_no_POL       equ *
1325                        *
1326                        * SCG
1327                        *
1328    00000558 0810 0007                  btst    #7,(a0)                 SCG interrupt?
1329    0000055C 6708                       beq.s   di_no_SCG               branch if not
1330    0000055E 6100 0008                  bsr     di_log                  else do normal logging
1331    00000562 0894 0007                  bclr    #7,(a4)                 mask off the interrupt
1332             0000 0566  di_no_SCG       equ *
1333    00000566 4E75                       rts
1334
1335                        *
1336                        * di_log    mark that an isr condition is pending
1337                        *
1338    00000568 08EA 0001  di_log          bset    #1,flags(a2)            set condition
                0039
1339    0000056E 4E75                       rts
1340
```



```
1343                        ********************************************************************************
1344                        *
1345                        *       di_tfr
1346                        *
1347                        *         driver call for execution of tfr statement
1348                        *
1349                        *         entry:  conditions other than normal a1,a2 are:
1350                        *                   a3.l = pointer to transfer information
1351                        *
1352                        ********************************************************************************
1353
1354    00000570 4EB9 0000  di_tfr          jsr     check_tfr               wait for tfr to finish (timed)
                0000
1355
1356    00000576 4A2B 000A                  tst.b   t_bw_off(a3)            don't allow word transfers
1357    0000057A 6600 FC62                  bne     di_noword
1358
1359    0000057E 202B 0010                  move.l  tcnt_off(a3),d0         get count
1360
1361    00000582 7200                       moveq   #0,d1
1362    00000584 122B 0009                  move.b  tusr_off(a3),d1         user specified transfer type
1363    00000588 D241                       add     d1,d1                   two bytes per table entry
1364    0000058A 4EB9 0000                  jsr     testdma                 DMA available?
                0000
1365    00000590 6704                       beq.s   di_nodma                branch if not; use the first half of table
1366    00000592 D27C 0014                  add     #20,d1                  otherwise use the second half of table
1367    00000596 323B 1006  di_nodma        move    tfr_table(d1),d1        routine offset
1368    0000059A 4EFB 1002                  jmp     tfr_table(d1)           jump to the appropriate routine
1369
1370                        *
1371                        * transfer jump table
1372                        *
1373                        *                                                 --------------------    DMA uninstalled or unavailable
1374    0000059E FC38       tfr_table       dc      hterr_b-tfr_table       serial  interrupt
1375    000005A0 FC3C                       dc      hterr_d-tfr_table       serial  DMA
1376    000005A2 00D2                       dc      di_t_fhs-tfr_table      serial  fhs
1377    000005A4 00D2                       dc      di_t_fhs-tfr_table      serial  fastest
1378    000005A6 FC38                       dc      hterr_b-tfr_table       serial  overlap
1379                        *                                               --------------------
1380    000005A8 008E                       dc      di_t_int-tfr_table      overlap interrupt
1381    000005AA FC3C                       dc      hterr_d-tfr_table       overlap DMA
1382    000005AC 00C8                       dc      di_t_bst-tfr_table      overlap fhs
1383    000005AE 00C8                       dc      di_t_bst-tfr_table      overlap fastest
1384    000005B0 008E                       dc      di_t_int-tfr_table      overlap overlap
1385                        *                                               --------------------    DMA available
1386    000005B2 FC38                       dc      hterr_b-tfr_table       serial  interrupt
1387    000005B4 0028                       dc      di_t_DMA-tfr_table      serial  DMA
1388    000005B6 00D2                       dc      di_t_fhs-tfr_table      serial  fhs
1389    000005B8 0028                       dc      di_t_DMA-tfr_table      serial  fastest
1390    000005BA FC38                       dc      hterr_b-tfr_table       serial  overlap
1391                        *                                               --------------------
1392    000005BC 008E                       dc      di_t_int-tfr_table      overlap interrupt
1393    000005BE 0028                       dc      di_t_DMA-tfr_table      overlap DMA
1394    000005C0 00C8                       dc      di_t_bst-tfr_table      overlap fhs
1395    000005C2 0028                       dc      di_t_DMA-tfr_table      overlap fastest
1396    000005C4 0028                       dc      di_t_DMA-tfr_table      overlap overlap
```

```
1398                            *
1399                            * transfer DMA
1400                            *
1401    000005C6 177C 0002 di_t_DMA        move.b  #tt_DMA,tact_off(a3)    set tfr type to DMA
                  0007
1402
1403    000005CC 2200                      move.l  d0,d1                   total count
1404    000005CE 5381                      subq.l  #1,d1                   total count minus 1
1405    000005D0 7000                      moveq   #0,d0
1406    000005D2 3001                      move.w  d1,d0                   first burst count minus 1
1407    000005D4 9280                      sub.l   d0,d1                   remaining count (multiple of $10000)
1408    000005D6 2541 003C                 move.l  d1,DMA_count(a2)        remember it
1409    000005DA 5280                      addq.l  #1,d0                   burst count [1..65536]
1410
1411    000005DC 4229 0003                 clr.b   cardcontrol(a1)         disable card interrupts
1412    000005E0 4EB9 0000                 jsr     getdma                  get a DMA channel
                  0000
1413    000005E6 0882 0000                 bclr    #0,d2                   always clear the DMA interrupt enable bit
1414    000005EA 4A2B 0030                 tst.b   t_DMApri(a3)            DMA priority requested?
1415    000005EE 6704                      beq.s   di_arm_DMA              branch if not
1416    000005F0 08C2 0003                 bset    #3,d2                   otherwise set the DMA priority bit
1417    000005F4 3882           di_arm_DMA move    d2,(a4)                 arm the channel
1418    000005F6 254C 0040                 move.l  a4,DMA_arm_addr(a2)     remember the DMA channel arm address
1419    000005FA 3542 0044                 move    d2,DMA_arm_word(a2)     remember the DMA channel arm word
1420
1421    000005FE 49F8 0000                 lea     0,a4                    no DMA termination routine (no DMA interrupt enabled)
1422    00000602 4EB9 0000                 jsr     DMA_stbsy               set DMA channel, buffer busy
                  0000
1423
1424    00000608 4A2B 000D                 tst.b   tdir_off(a3)            test the transfer direction
1425    0000060C 6614                      bne.s   di_tod                  branch if output
1426
1427                            *
1428                            * transfer input DMA:
1429                            *
1430    0000060E 08E9 0001 di_tid          bset    #1,intmask(a1)          interrupt on DMA completion or fifo idle
                  0013
1431    00000614 08A9 0001                 bclr    #1,control(a1)          select the inbound fifo for DMA
                  0019
1432    0000061A D081                      add.l   d1,d0                   recompute the total count again
1433    0000061C 6100 005C                 bsr     input_tfr_en            enable the transfer
1434    00000620 6024                      bra.s   di_DMA_en               enable the card & wait if serial
1435
1436
1437                            *
1438                            * transfer output DMA:
1439                            *
1440    00000622 08E9 0001 di_tod          bset    #1,control(a1)          select the outbound fifo for DMA
                  0019
1441    00000628 08E9 0007                 bset    #7,control(a1)          assume no EOI tag (8-bit mode)
                  0019
1442    0000062E 4229 0017                 clr.b   status(a1)              clear the high-order bits
1443    00000632 4A81                      tst.l   d1                      remaining count 0?
1444    00000634 660C                      bne.s   di_tod_1                branch if not
1445    00000636 4A2B 000B                 tst.b   tend_off(a3)            EOI tag set?
1446    0000063A 6706                      beq.s   di_tod_1                branch if not, otherwise...
```



```
1447    0000063C 08A9 0007                 bclr    #7,control(a1)          clear 8-bit mode (tag last byte with EOI)
                  0019
1448    00000642 08C3 0003 di_tod_1        bset    #3,d3                   set the card's output DMA enable bit
1449    00000646 1343 0003 di_DMA_en       move.b  d3,cardcontrol(a1)      enable card interrupts & DMA
1450
1451    0000064A 0C2B 0005 di_DMA_w        cmpi.b  #5,tusr_off(a3)         overlap transfer?
                  0009
1452    00000650 6C08                      bge.s   di_DMA_w2               branch if so
1453
1454    00000652 0C2B 00FF di_DMA_w1       cmpi.b  #255,t_sc_off(a3)       if not then wait till done
                  0005
1455    00000658 66F8                      bne.s   di_DMA_w1
1456
1457    0000065A 4E75           di_DMA_w2  rts
1458
```

```
1460                        *
1461                        * transfer interrupt
1462                        *
1463    0000065C 177C 0001 di_t_int        move.b  #tt_int,tact_off(a3)    set tfr type to interrupt
                0007
1464    00000662 6000 FB72                 bra     hterr_b
1465
1466
1467                        *
1468                        * transfer burst (intr on 1st byte fhs on rest)
1469                        *
1470    00000666 177C 0003 di_t_bst        move.b  #tt_burst,tact_off(a3)  set tfr type to burst
                0007
1471    0000066C 6000 FB68                 bra     hterr_b
1472
1473
1474                        *
1475                        * transfer fast handshake
1476                        *
1477    00000670 177C 0004 di_t_fhs        move.b  #tt_fhs,tact_off(a3)    set tfr type to fhs
                0007
1478    00000676 6000 FB5E                 bra     hterr_b
1479
```



```
1481                        *
1482                        * input transfer enable
1483                        *
1484    0000067A 0829 0004 input_tfr_en    btst    #4,status(a1)           active controller?
                0017
1485    00000680 6714                      beq.s   te_rts                  branch if not
1486
1487    00000682 B0BC 0000                 cmp.l   #256,d0                 count low enough for counted transfer?
                0100
1488    00000688 6F0E                      ble.s   te_ect                  branch if so
1489
1490    0000068A 137C 00C0                 move.b  #$C0,status(a1)
                0017
1491    00000690 137C 0000                 move.b  #$00,fifo(a1)           uncounted transfer enable
                0015
1492    00000696 4E75       te_rts         rts
1493
1494    00000698 137C 0080 te_ect          move.b  #$80,status(a1)         inhibit LF detection
                0017
1495    0000069E 1340 0015                 move.b  d0,fifo(a1)             counted transfer enable
1496    000006A2 4E75                      rts
1497
1498
1499                        *
1500                        * input transfer term
1501                        *
1502    000006A4 0829 0004 input_tfr_term  btst    #4,status(a1)           active controller?
                0017
1503    000006AA 674A                      beq.s   tt_rts                  branch if not
1504
1505    000006AC 0829 0001                 btst    #1,intreg(a1)           fifo idle?
                0011
1506    000006B2 6628                      bne.s   tt_empty_fifo           branch if so
1507                        *
1508                        * (input) transfer enable byte still active - must init outbound fifo
1509                        *
1510    000006B4 0829 0003                 btst    #3,status(a1)           system controller?
                0017
1511    000006BA 671A                      beq.s   tt_non_sys              branch if not
1512
1513    000006BC 0029 0011                 ori.b    #$11,control(a1)       assert IFC & init outbound fifo
                0019
1514                        *   tttt jws   moveq   #80,d0                  must hold IFC at least 100us...
1515                        *   tttt jws   dbra    d0,*                    to satisfy the bus standard
1516    000006C2 2F3C 0000                 move.l  #100,-(sp)              use timer routine  tttt jws
                0064
1517    000006C8 4EB9 0000                 jsr     delay_timer             for this delay     tttt jws
                0000
1518    000006CE 08A9 0004                 bclr    #4,control(a1)          release IFC
                0019
1519    000006D4 6006                      bra.s   tt_empty_fifo           go empty the inbound fifo
1520
1521    000006D6 08E9 0000 tt_non_sys      bset    #0,control(a1)          init outbound fifo; TCA like this is hazardous!!!
                0019
1522                        *
1523                        * empty inbound fifo
1524                        *
```

```
 1525    000006DC 08E9 0002 tt_empty_fifo   bset    #2,intmask(a1)          unmask fifo byte condition
                    0013
 1526    000006E2 0829 0002                 btst    #2,intreg(a1)           fifo byte?
                    0011
 1527    000006E8 6706                      beq.s   tt_fifo_empty           branch if not
 1528    000006EA 4A29 0015                 tst.b   fifo(a1)               otherwise, discard it and...
 1529    000006EE 60EC                      bra     tt_empty_fifo           go check for more
 1530
 1531    000006F0 08A9 0002 tt_fifo_empty   bclr    #2,irtmask(a1)          re-mask fifo byte condition
                    0013
 1532
 1533    000006F6 4E75      tt_rts          rts
 1534
 1535
 1536                                       end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ABORT_IO	ABS	146	00000015
CHECK_TFR	ABS	148	0000001B
CHECK_TIMER	ABS	149	0000001E
DELAY_TIMER	ABS	150	00000021
DMA_STBSY	ABS	144	00000010
DROPDMA	ABS	137	00000002
GETDMA	ABS	138	00000004
IODECLARATIONS	ABS	295	00000024
ITXFR	ABS	145	00000013
LOGEOT	ABS	141	0000000A
LOGINT	ABS	140	00000008
STBSY	ABS	142	0000000C
STCLR	ABS	143	0000000E
SYSGLOBALS	ABS	296	00000028
TESTDMA	ABS	139	00000006
WAIT_TFR	ABS	147	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	344		00000034
ADDRESS	ABS	572		0000001B
AVAIL_OFF	ABS	235		00000034
BADTMO	ABS	313		0000000B
BAD_RDS	ABS	321		00000013
BAD_SCT	ABS	322		00000014
BUFI_OFF	ABS	226		00000024
BUFO_OFF	ABS	227		00000028
BUF_BUSY	ABS	311		00000009
CARDCONTROL	ABS	564		00000003
CARDID	ABS	561		00000001
CARDLATCH	ABS	565		00000007
CARDRESET	ABS	562		00000001
CARDSTATUS	ABS	563		00000003
CCR	STREG	0		00000005
CLEAR_TABLE	REL	1063		000003D8
CONTROL	ABS	571		00000019
CRD_DWN	ABS	323		00000015
C_ADR	ABS	225		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      589                        00000014
DFC               STREG      0                        00000008
DI_ARM_DMA        REL     1417                        000005F4
DI_BUF            REL     1246                        00000502
DI_BUFI           REL     1253                        00000508
DI_BUFO           REL     1258                        0000050A
DI_CLR            REL     1057                        000003C6
DI_DMA_EN         REL     1449                        00000646
DI_DMA_W          REL     1451                        0000064A
DI_DMA_W1         REL     1454                        00000652
DI_DMA_W2         REL     1457                        0000065A
DI_DMYRTS         REL     1079                        000003F8
DI_EOI            REL     1043                        000003C0
DI_FRI            REL     1282                        00000522
DI_FRW            REL     1274                        00000518
DI_FRWO           REL     1287                        00000524
DI_IFC            REL     1024                        00000394
DI_INIT           REL      607                        0000011E
DI_INIT_S         REL      625                        00000128
DI_ISR            REL     1147                        0000041E
DI_ISR1           REL     1299                        00000526
DI_ISRDMA         REL     1166                        00000456
DI_ISR_EX         REL     1160                        00000454
DI_LOCAL          REL     1076                        000003E8
DI_LOG            REL     1338                        00000568
DI_LSTERR         REL      728                        000001E2
DI_NODMA          REL     1367                        00000596
DI_NOTACTL        REL      713                        000001CE
DI_NOTSCTL        REL      716                        000001D2
DI_NOWORD         REL      725                        000001DE
DI_NO_OCL         REL     1308                        0000053C
DI_NO_POL         REL     1324                        00000558
DI_NO_SCG         REL     1332                        00000566
DI_NO_SRQ         REL     1316                        0000054A
DI_P_POLL         REL      969                        0000032A
DI_RBOUT          REL     1120                        000003FE
DI_RDB            REL      661                        0000017A
DI_RDB1           REL      668                        00000194
DI_RDS            REL      826                        00000274
DI_REN            REL     1015                        00000382
DI_RQS            REL      936                        000002FA
DI_RQS2           REL      942                        0000030E
DI_SCBSY          REL      707                        000001C6
DI_SC_ERR         REL      710                        000001CA
DI_SET            REL      996                        00000360
DI_TEST           REL     1102                        000003FA
DI_TFR            REL     1354                        00000570
DI_TID            REL     1430                        0000060E
DI_TI_TERM        REL     1263                        0000050A
DI_TLKERR         REL      731                        000001E6
DI_TMO            REL      734                        000001EA
DI_TOD            REL     1440                        00000622
DI_TOD_1          REL     1448                        00000642
```



```
DI_TO_TERM        REL     1265                        0000050A
DI_T_BST          REL     1470                        00000666
DI_T_DMA          REL     1401                        000005C6
DI_T_FHS          REL     1477                        00000670
DI_T_INT          REL     1463                        0000065C
DI_WAIT_FB        REL      754                        00000204
DI_WAIT_FI        REL      751                        00000200
DI_WFC            REL      759                        00000208
DI_WFC_DONE       REL      797                        0000024A
DI_WFC_INFINITE   REL      792                        00000246
DI_WFC_QUICK      REL      769                        00000218
DI_WFC_TEXIT      REL      813                        00000270
DI_WFC_TIMED      REL      781                        00000232
DI_WFC_TIMER      REL      803                        00000256
DI_WFC_TLOOP      REL      805                        0000025C
DI_WTB            REL      689                        000001A8
DI_WTC            REL      906                        000002D4
DI_WTC_RST        REL      923                        000002E4
DMA_ARM_ADDR      ABS      384                        00000040
DMA_ARM_WORD      ABS      385                        00000044
DMA_COUNT         ABS      383                        0000003C
EIRB_OFF          ABS      228                        0000002C
EOD_SEEN          ABS      324                        00000016
EOI_IN            ABS      345                        00000035
EOI_OUT           ABS      346                        00000036
ESC_CODE          ABS      332 SYSGLOBALS      +      FFFFFFFE
ESC_ERR           REL      738                        000001EC
EXTDI_EDI_CLR     REL      519                        000000E2
EXTDI_EDI_END     REL      543                        00000108
EXTDI_EDI_INIT    REL      383                        00000002
EXTDI_EDI_ISR     REL      392                        00000010
EXTDI_EDI_PPOLL   REL      497                        000000BE
EXTDI_EDI_RDB     REL      401                        0000001E
EXTDI_EDI_RDS     REL      452                        00000076
EXTDI_EDI_RDW     REL      424                        00000042
EXTDI_EDI_SEND    REL      487                        000000AE
EXTDI_EDI_SET     REL      509                        000000D2
EXTDI_EDI_TEST    REL      530                        000000F2
EXTDI_EDI_TFR     REL      477                        0000009E
EXTDI_EDI_WTB     REL      413                        00000032
EXTDI_EDI_WTC     REL      465                        0000008C
EXTDI_EDI_WTW     REL      438                        0000005C
EXTDI_EXTDI       REL      378                        00000000
FIFO              ABS      569                        00000015
FLAGS             ABS      348                        00000039
GET               ABS      586                        00000008
GTL               ABS      583                        00000001
HPL_WTC1          REL      925                        000002E8
HPL_WTC2          REL      930                        000002F8
HPL_WTC4          REL      944                        00000314
HPL_WTC5          REL      950                        00000324
HTERR_B           REL      719                        000001D6
HTERR_D           REL      722                        000001DA
HWTCTRL           REL      912                        000002E0
H_ISR_PM          ABS      224                        0000001C
H_ISR_PR          ABS      222                        00000014
H_ISR_SL          ABS      223                        00000018
```

```
INPUT_TFR_EN    REL     1484                        0000067A
INPUT_TFR_TERM  REL     1502                        000006A4
INTMASK         ABS      568                        00000013
INTREG          ABS      567                        00000011
IOE_ERROR       ABS      327                        FFFFFFE6
IOE_RSLT        ABS      329 IODECLARATIONS  +      FFFFFFBE
IOE_SC          ABS      330 IODECLARATIONS  +      FFFFFFBA
IO_MISC         ABS      325                        00000017
IO_SC           ABS      229                        0000002D
ISRDMA_IN_TERM  REL     1189                        00000486
ISRDMA_OUT      REL     1197                        00000492
ISRDMA_RESTART  REL     1217                        000004BE
ISRDMA_TERM     REL     1228                        000004D8
ISRDMA_UNARMED  REL     1186                        00000480
ISR_END         REL     1157                        00000446
ISR_ENTRY       ABS      220                        00000000
LLO             ABS      588                        00000011
MA              ABS      234                        00000033
MA_W            ABS      233                        00000032
NOT_HPIB        ABS      304                        00000002
NOT_LSTN        ABS      318                        00000010
NOT_TALK        ABS      317                        0000000F
NO_ACTL         ABS      305                        00000003
NO_CARD         ABS      303                        00000001
NO_DATA         ABS      308                        00000006
NO_DMA          ABS      315                        0000000D
NO_DRV          ABS      314                        0000000C
NO_DVC          ABS      306                        00000004
NO_SCTL         ABS      320                        00000012
NO_SPACE        ABS      307                        00000005
NO_WORD         ABS      316                        0000000E
PPC             ABS      585                        00000005
PPD             ABS      596                        00000070
PPE             ABS      595                        00000060
PPOLLMSK        ABS      360                        0000003A
PPOLMASK        ABS      573                        0000001D
PPOLSENSE       ABS      574                        0000001F
PPU             ABS      590                        00000015
RCVR_BLK        ABS      333 SYSGLOBALS      +      FFFFFFF6
RDS_0           REL      859                        0000029C
RDS_1           REL      845                        00000296
RDS_2           REL      846                        00000296
RDS_3           REL      866                        000002A0
RDS_4           REL      847                        00000296
RDS_5           REL      848                        00000296
RDS_6           REL      879                        000002B4
RDS_7           REL      849                        00000296
RDS_8           REL      850                        00000296
RDS_ERR         REL      852                        00000296
RDS_TABLE       REL      835                        00000284
SC_BUSY         ABS      310                        00000008
SDC             ABS      584                        00000004
SET_TABLE       REL     1002                        00000372
SFC             STREG      0                        00000009
SP              AREG       0                        00000007
SPD             ABS      592                        00000019
SPE             ABS      591                        00000018
```



```
SR              STREG      0                        00000006
STATUS          ABS      570                        00000017
SYSFLAG2        ABS      336                        FFFFFED8
TACT_OFF        ABS      250                        00000007
TBSZ_OFF        ABS      273                        00000018
TBUF_OFF        ABS      272                        00000014
TCHR_OFF        ABS      269                        0000000E
TCNTERR         ABS      312                        0000000A
TCNT_OFF        ABS      271                        00000010
TCT             ABS      587                        00000009
TDIR_OFF        ABS      267                        0000000D
TEMP_OFF        ABS      274                        0000001C
TEND_OFF        ABS      265                        0000000B
TE_ECT          REL     1494                        00000698
TE_RTS          REL     1492                        00000696
TFIL_OFF        ABS      275                        00000020
TFR_ERR         ABS      309                        00000007
TFR_TABLE       REL     1374                        0000059E
TIMEOUT         ABS      230                        0000002E
TIMER_PRESENT   ABS      335                        00000001
TMO_ERR         ABS      319                        00000011
TTMP_OFF        ABS      248                        00000000
TT_BURST        ABS      286                        00000003
TT_DMA          ABS      285                        00000002
TT_EMPTY_FIFO   REL     1525                        000006DC
TT_FHS          ABS      287                        00000004
TT_FIFO_EMPTY   REL     1531                        000006F0
TT_INT          ABS      284                        00000001
TT_NON_SYS      REL     1521                        000006D6
TT_RTS          REL     1533                        000006F6
TUSR_OFF        ABS      251                        00000009
T_BW_OFF        ABS      263                        0000000A
T_DMAPRI        ABS      280                        00000030
T_PM_OFF        ABS      279                        0000002C
T_PR_OFF        ABS      276                        00000024
T_SC_OFF        ABS      249                        00000005
T_SL_OFF        ABS      278                        00000028
UNL             ABS      593                        0000003F
UNT             ABS      594                        0000005F
USER_ISR        ABS      221                        00000014
USP             STREG      0                        00000007
VBR             STREG      0                        0000000A
```

Description

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

```
PAGE   1 [3.0] 12/26/84 21:38:40  ASSEMBLY OF DRVASM.TEXT                                   *** File name: DRVASM ***

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        function shifted_left(value: integer; n: shortint): integer;
   21                            src        function shifted_right(value: integer; n: shortint): integer;
   22                            src        function mod_power_of_2(value: integer; n: shortint): integer;
   23                            src
   24                            src    end; {drvasm}
   25
   26
   27                           *
   28                           * def's
   29                           *
   30                                       def     drvasm_drvasm
   31                                       def     drvasm_test_and_toggle
   32                                       def     drvasm_eor
   33                                       def     drvasm_shifted_left
   34                                       def     drvasm_shifted_right
   35                                       def     drvasm_mod_power_of_2
   36
   37
   38                           *
   39                           * module intialization routine
   40                           *
   41    00000000 4E75          drvasm_drvasm   rts
   42
   43
   44                           *
   45                           * test and toggle  -  semaphore manipulation function
   46                           *
   47              0000 0002 drvasm_test_and_toggle equ *
   48    00000002 205F                          movea.l (sp)+,a0        pop the return address
   49    00000004 225F                          movea.l (sp)+,a1        pop the var parameter address
   50    00000006 0851 0000                     bchg    #0,(a1)         test and toggle the semaphore
   51    0000000A 56C0                          sne     d0              remember the previous state
   52    0000000C 4400                          neg.b   d0              form a legal PASCAL boolean
   53    0000000E 1E80                          move.b  d0,(sp)         set the return variable
   54    00000010 4ED0                          jmp     (a0)            return
   55
```

```
PAGE   2 [3.0] 12/26/84 21:38:40  ASSEMBLY OF DRVASM.TEXT                                   *** File name: DRVASM ***

   57                           *
   58                           * exclusive or  -  error correction procedure
   59                           *
   60    00000012 205F          drvasm_eor      movea.l (sp)+,a0        pop the return address
   61    00000014 225F                          movea.l (sp)+,a1        pop the bufptr
   62    00000016 101F                          move.b  (sp)+,d0        pop the correction character
   63    00000018 B111                          eor.b   d0,(a1)         do it to it
   64    0000001A 4ED0                          jmp     (a0)            return
   65
   66
   67                           *
   68                           * shift left n places
   69                           *
   70              0000 001C drvasm_shifted_left equ *
   71    0000001C 205F                          movea.l (sp)+,a0        pop the return address
   72    0000001E 321F                          move    (sp)+,d1        pop the shift count
   73    00000020 201F                          move.l  (sp)+,d0        pop the operand
   74    00000022 E3A0                          asl.l   d1,d0           do it to it
   75    00000024 2E80                          move.l  d0,(sp)         set the return value
   76    00000026 4ED0                          jmp     (a0)            return
   77
   78
   79                           *
   80                           * shift right n places
   81                           *
   82              0000 0028 drvasm_shifted_right equ *
   83    00000028 205F                          movea.l (sp)+,a0        pop the return address
   84    0000002A 321F                          move    (sp)+,d1        pop the shift count
   85    0000002C 201F                          move.l  (sp)+,d0        pop the operand
   86    0000002E E2A0                          asr.l   d1,d0           do it to it
   87    00000030 2E80                          move.l  d0,(sp)         set the return value
   88    00000032 4ED0                          jmp     (a0)            return
   89
   90
   91                           *
   92                           * take a mod power of 2
   93                           *
   94              0000 0034 drvasm_mod_power_of_2 equ *
   95    00000034 205F                          movea.l (sp)+,a0        pop the return address
   96    00000036 321F                          move    (sp)+,d1        pop n
   97    00000038 70FF                          moveq   #-1,d0          start with all ones
   98    0000003A E3A0                          asl.l   d1,d0           shift in n zeros
   99    0000003C 4680                          not.l   d0              invert the sense
  100    0000003E C09F                          and.l   (sp)+,d0        pop and mask the operand
  101    00000040 2E80                          move.l  d0,(sp)         set the return value
  102    00000042 4ED0                          jmp     (a0)            return
  103
  104                                           end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

EVALGVR

Description

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 a6
   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 1380                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
```

```
  222    00000186 4E75              rts
  223    00000188 DDA9 0026 addreloc add.l  relocdelta(modptr),gvalue
  224    0000018C 6300 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
```

Description

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 (which employ the above procedures) may have unpredictable results if the direction is opposite that which is appropriate.

```
PAGE   1 [3.0] 12/26/84 21:00:23  ASSEMBLY OF FASTMOVE.TEXT                                  *** File name: FASTMOVE ***

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
```

```
PAGE   2 [3.0] 12/26/84 21:00:23  ASSEMBLY OF FASTMOVE.TEXT                                  *** File name: FASTMOVE ***

   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 [3.0] 12/26/84 21:00:23  ASSEMBLY OF FASTMOVE.TEXT                                  *** File name: FASTMOVE ***

  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
```

Description

FMINIT is the initialization driver for the 9885 disc.

Usage

FMINIT is used by MEDIAINIT for 9885 discs.

```
PAGE   1 [3.0] 12/26/84 22:57:36  ASSEMBLY OF FMINIT.TEXT                                   *** File name: FMINIT ***

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 [3.0] 12/26/84 22:57:36 FMINIT - 9885 INITIALIZATION ROUTINE                   *** File name: FMINIT ***

  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 0398          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 0390          bsr     f85wo                   issue it
00000098 6100 0396          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 0382          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 6156           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 02F6              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 02EE              bsr     f85wo                   issue it

0000013A 6100 0194              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 7028                   moveq   #40,d0          wait a bit for card to reset     JS 8/19/83
00000152 51C8 FFFE              dbra    d0,*                                             JS 8/19/83

00000156 7011                   moveq   #ior_initfailed,d0      ioresult in case of excessive rejected tracks
00000158 0C2E 003F              cmpi.b  #63,v85gtc(a6)          did we get enough good tracks?
         FFFC
0000015E 6D00 FEDE              blt     ioresc                  escape if not

00000162 4E5E                   unlk    a6                      remove our stack frame
00000164 205F                   movea.l (sp)+,a0                pop off the return address
00000166 508F                   addq.l  #8,sp                   pop off the parameters
00000168 4ED0                   jmp     (a0)                    return


                        *
                        * test patterns (the first is written by the format command!)
                        *                       1               2
                        *       dc      $C6C6   1100011011000110110001101100011
0000016A 6363           f85ptrn dc      $6363   0110001101100011011000110110001
0000016C DB6D                   dc      $DB6D   1101101101101101101101101101101
0000016E 8888                   dc      $8888   1000100010001000100010001000100
00000170 0000                   dc      $0000   (final sector contents)
```



```
                        *
                        * routine to format and test one track
                        *
00000172 102E FFFC f85fmt  move.b  v85gtc(a6),d0           good track count
00000176 B03C 003F              cmp.b   #63,d0                  already have enough good tracks?
0000017A 6C00 011E              bge     f85mtd                  branch if so (don't leave "extra" tracks)

0000017E D02E FFFD              add.b   v85btc(a6),d0           compute current physical track number
00000182 B03C 0042              cmp.b   #66,d0                  out past the 9885's "supported" range?
00000186 6E00 0112              bgt     f85mtd                  branch if so (don't leave "extra" tracks)

0000018A 6100 0296              bsr     f85lgin                 give the password
0000018E 302E FFF6              move    v85scmd(a6),d0          basic command skeleton
00000192 EA58                   ror     #5,d0
00000194 802E FFFC              or.b    v85gtc(a6),d0           current logical track number
00000198 EB58                   rol     #5,d0
0000019A 807C 001E              or      #30,d0                  format command for this track

0000019E 6100 027A              bsr     f85wf                   wait for the flag
000001A2 3940 0004              move    d0,4(a4)                place the command in the output buffer
000001A6 4E4B                   trap    #11                                                  scs
                        * scs   move    sr,-(sp)                prepare to disable interrupts
000001A8 007C 2700              ori     #$2700,sr               disable interrupts
                        ************************ interrupts disabled *************************
000001AC 1880                   move.b  d0,(a4)                 set the peripheral control flag

000001AE 721D                   moveq   #30-1,d1                initialize the loop counter
000001B0 206E FFEC              movea.l v85lrso(a6),a0          logical record spiral offset
000001B4 7000                   moveq   #0,d0                   need upper byte cleared

000001B6 1018           f85flb  move.b  (a0)+,d0                next logical record number to send

000001B8 6100 024C              bsr     f85cswf                 check status and wait for the flag
000001BC 600A                     bra.s f85fle                  ret 1; terminate & check status

000001BE 3940 0004              move    d0,4(a4)                place the lrn in the output buffer
000001C2 1880                   move.b  d0,(a4)                 set the peripheral control flag

000001C4 51C9 FFF0              dbra    d1,f85flb               loop until all 30 lrn's sent

000001C8 46DF           f85fle  move    (sp)+,sr                re-enable interrupts
                        ************************ interrupts re-enabled ************************
000001CA 6100 0104              bsr     f85fst                  fetch status
000001CE C07C FFF8              and     #$FFF8,d0               strip the drive & disc change bits
000001D2 B07C 0040              cmp     #$0040,d0               format complete, no errors?
000001D6 6600 FE72              bne     f85die                  if not, decode and issue error
                        *
                        * prepare for reading and writing under i/o control
                        *
000001DA 47FA FF8E              lea     f85ptrn,a3              first test pattern address

000001DE 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
                        *
000001E4 206E FFEC f85vwl  movea.l v85lrso(a6),a0          logical record spiral offset
000001E8 2D48 FFE8         move.l  a0,v85lrp(a6)           init the logical record pointer

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

000001F2 7000              moveq   #0,d0                   clear the upper byte
000001F4 102E FFFC         move.b  v85gtc(a6),d0           this track's logical track#
000001F8 C0FC 001E         mulu    #30,d0                  this track's logical record 0
000001FC 3D40 FFF4         move    d0,v85lr0(a6)           save it

00000200 206E FFE8 f85v1   movea.l v85lrp(a6),a0           logical record pointer
00000204 7000              moveq   #0,d0                   clear upper byte and word
00000206 1018              move.b  (a0)+,d0                logical record to verify next
00000208 D06E FFF4         add     v85lr0(a6),d0           this track's logical record 0
0000020C 2D48 FFE8         move.l  a0,v85lrp(a6)           updated logical record pointer
00000210 3D40 FFF2         move    d0,v85frec(a6)          record to verify

00000214 51EE FFFB         sf      v85rwf(a6)              set read/write flag to read
00000218 6100 00D0         bsr     f85xfr                  do the transfer
0000021C 6042                bra.s f85ferr                 ret 1; some error occurred

0000021E 532E FFFA         subq.b  #1,v85rc(a6)            decrement the record counter
00000222 6EDC              bgt     f85v1                   loop til all 30 records are verified

                        *
                        * write under i/o control
                        *
00000224 3013              move    (a3),d0                 test pattern
00000226 4840              swap    d0
00000228 301B              move    (a3)+,d0                need it in upper & lower words
0000022A 41EE FEAA         lea     v85buf(a6),a0           first word address
0000022E 723F              moveq   #64-1,d1                initialize the loop counter
00000230 20C0              move.l  d0,(a0)+                write four bytes
00000232 51C9 FFFC         dbra    d1,*-2                  loop until done

00000236 206E FFEC         movea.l v85lrso(a6),a0          logical record spiral offset
0000023A 2D48 FFE8         move.l  a0,v85lrp(a6)           logical record pointer

0000023E 1D7C 001E         move.b  #30,v85rc(a6)           init the record count
         FFFA
```



```
00000244 206E FFE8 f85w1   movea.l v85lrp(a6),a0           logical record pointer
00000248 7000              moveq   #0,d0                   clear upper byte and word
0000024A 1018              move.b  (a0)+,d0                next record to be written
0000024C D06E FFF4         add     v85lr0(a6),d0           this track's logical record 0
00000250 2D48 FFE8         move.l  a0,v85lrp(a6)           updated logical record pointer
00000254 3D40 FFF2         move    d0,v85frec(a6)          record number to write

00000258 50EE FFFB         st      v85rwf(a6)              set the read/write flag to write
0000025C 6100 008C         bsr     f85xfr                  do the transfer
00000260 6020      f85ferr   bra.s f85err                  ret 1; some error occurred

00000262 532E FFFA         subq.b  #1,v85rc(a6)            decrement the record count
00000266 6EDC              bgt     f85w1                   loop till all 30 records are written


00000268 532E FFFE         subq.b  #1,v85patc(a6)          decrement the pattern count
0000026C 6E00 FF76         bgt     f85vwl                  loop until done

00000270 522E FFFC         addq.b  #1,v85gtc(a6)           increment the good track count

00000274 701A              moveq   #26,d0
00000276 906E 0008         sub     int1ve(a6),d0
0000027A 906E FFF0         sub     v85r29o(a6),d0          (offset=offset-inter-r29o+26)
0000027E 48C0              ext.l   d0                      need the offset long
00000280 6038              bra.s   f85bof                  bump the logical record offset


                        *
                        * an error occurred in verifying or writing: is it "ok"?
                        *
00000282 102E FFF8 f85err  move.b  v85stat(a6),d0          the upper byte of the error code word

00000286 5B00              subq.b  #5,d0                   id error?
00000288 6710              beq.s   f85mtd                  branch if so
0000028A 5300              subq.b  #6-5,d0                 track error?
0000028C 670C              beq.s   f85mtd                  branch if so
0000028E 5300              subq.b  #7-6,d0                 crc error?
00000290 6708              beq.s   f85mtd                  branch if so
00000292 5500              subq.b  #9-7,d0                 verify error?
00000294 6704              beq.s   f85mtd                  branch if so

00000296 6000 FDB2         bra     f85die                  take the error exit
```

```
                        *
                        * mark the (bad) track defective (invisible)
                        *
0000029A 6100 0186 f85mtd  bsr     f851gin                 give the password

0000029E 302E FFF6         move    v85scmd(a6),d0          basic command skeleton
000002A2 807C 0FBF         or      #$0FBF,d0               form a mark track defective cmd
000002A6 6100 017E         bsr     f85wo                   issue it

000002AA 6124              bsr.s   f85fst                  fetch status
000002AC C07C FFFC         and     #$FFFC,d0               strip off the drive bits
000002B0 6600 FD98         bne     f85die                  branch if any errors

000002B4 522E FFFD         addq.b  #1,v85btc(a6)           increment the bad track count

000002B8 70FC              moveq   #-4,d0                  logical record offset to be bumped by 4
                        *  bra.s   f85bof                  bump the offset

                        *
                        * subroutine to bump the logical record spiral offset
                        *
000002BA D0AE FFEC f85bof  add.l   v851rso(a6),d0          bump it
000002BE 2D40 FFEC         move.l  d0,v851rso(a6)          save it
000002C2 41EE FFAA         lea     v85ibuf(a6),a0          interleave buffer's FBA
000002C6 B088              cmp.l   a0,d0                   offset still in range?
000002C8 6C04              bge.s   *+6                     branch if so
000002CA 701E              moveq   #30,d0                  else put it in range
000002CC 60EC              bra.s   f85bof
000002CE 4E75              rts

                        *
                        * subroutine to fetch status
                        *
000002D0 6100 0134 f85fst  bsr     f85cswf                 check status and wait for the flag
000002D4 4E71                nop                           ret 1; error occurred

000002D6 6100 015A         bsr     f85winw                 word in; don't wait on the flag
000002DA 3D40 FFF8         move    d0,v85stat(a6)          save the status word for now

000002DE 7000              moveq   #0,d0
000002E0 6100 0144         bsr     f85wo                   complete the handshake with the '85

000002E4 302E FFF8         move    v85stat(a6),d0          load the status word
000002E8 4E75              rts
```



```
                        ************************************************************************
                        *                        routine to transfer one sector                *
                        ************************************************************************

                        *
                        * gain access to the dma channel
                        *
000002EA 247C 0050 f85xfr  movea.l #$500000,a2             DMA channel 0 address
         0000
000002F0 7005              moveq   #$05,d0                 gpio dma control byte: chan 0
                        *
                        * set up the gpio card
                        *
000002F2 1D40 FFFF         move.b  d0,v85gdcb(a6)          save the gpio dma control byte
000002F6 197C 0000         move.b  #0,3(a4)                clear enab, word, dmac1, dmac0
         0003
000002FC 197C 0000         move.b  #0,7(a4)                clear ctl1 & ctl0
         0007
                        *
                        * build and issue the seek command
                        *
00000302 6100 011E         bsr     f851gin                 give the password

00000306 302E FFF2         move    v85frec(a6),d0          first record number
0000030A 48C0              ext.l   d0                      make it long for division
0000030C 81FC 001E         divs    #30,d0                  split into track & sector #'s
00000310 4840              swap    d0                      track in upper; sector in lower
00000312 EA58              ror     #5,d0                   sector in upper bits of lower
00000314 EE98              ror.l   #7,d0                   track&sector in upper bits of lower
00000316 806E 000A         or      un(a6),d0               track&sector&00&unit in lower
0000031A E458              ror     #2,d0                   unit&track&sector&00
0000031C 807C 0003         or      #3,d0                   unit&track&sector&11
00000320 E458              ror     #2,d0                   11&unit&track&sector (finally!)
00000322 6100 0102         bsr     f85wo                   issue the seek command

                        *
                        * partially build the appropriate transfer command
                        *
00000326 6100 00FA         bsr     f851gin                 give the password

0000032A 302E 000A         move    un(a6),d0               unit number
0000032E E858              ror     #4,d0                   position for read/write/verify
00000330 5240              addq    #1,d0                   specify a record count of one

00000332 41EE FEAA         lea     v85buf(a6),a0           read/write buffer's FWA
00000336 2488              move.l  a0,(a2)                 write in the DMA channel's address register

00000338 727F              moveq   #127,d1                 #words-1 in a sector
0000033A 3541 0004         move    d1,4(a2)                write in the DMA channel's count register

0000033E 6100 00C6         bsr     f85cswf                 check status and wait for the flag
00000342 605A                bra.s f85_grs                 ret 1 - error; go read status

00000344 4A2E FFFB         tst.b   v85rwf(a6)              read/write flag
00000348 6B56              bmi.s   f85_wrt                 branch if a write
```

```
                        *
                        * section for verify (read)
                        *
0000034A 08C0 000E              bset    #14,d0                  make a verify command
0000034E 3940 0004              move    d0,4(a4)                place cmnd in the output buffer

00000352 4E4B                   trap    #11                                                     scs
                        * scs   move    sr,-(sp)                prepare to disable interrupts
00000354 007C 2700              ori     #$2700,sr               disable interrupts (except nmi)
                        ************************ interrupts disabled *************************
00000358 1880                   move.b  d0,(a4)                 set the peripheral control line

0000035A 6100 00AA              bsr     f85cswf                 check status and wait for the flag
0000035E 600C                     bra.s f85_rby                 ret 1; bypass because of error

00000360 397C 0000              move    #0,4(a4)                clear the output buffers
                 0004
00000366 4A6C 0004              tst     4(a4)                   set direction in w/ dummy read
0000036A 1880                   move.b  d0,(a4)                 set the peripheral control flag

0000036C 196E FFFF f85_rby      move.b  v85gdcb(a6),3(a4)       set the gpio dma control bits
                 0003
00000372 536A 0004              subq    #1,4(a2)                don't dma the last word in
00000376 6D06                   blt.s   f85_rei                 don't arm if no words to dma in
00000378 357C 0002              move    #$0002,6(a2)            arm the dma channel
                 0006
0000037E 46DF           f85_rei move    (sp)+,sr                re-enable interrupts
                        ************************ interrupts re-enabled ************************
00000380 41EE FFA8              lea     v85buf+254(a6),a0       transfer's last word address

00000384 082C 0003 f85_rwt      btst    #3,7(a4)                peripheral status set?
                 0007
0000038A 6648                   bne.s   f85_tdx                 if so, terminate dma xfer now!
0000038C 082A 0000              btst    #0,7(a2)                dma channel still armed?
                 0007
00000392 66F0                   bne     f85_rwt                 if so, loop

00000394 6100 0070              bsr     f85cswf                 check status and wait for the flag
00000398 603A                     bra.s f85_tdx                 ret 1 - error; bypass else hang
0000039A 30AC 0004              move    4(a4),(a0)              transfer the last word
0000039E 6034           f85_grs bra.s   f85_tdx                 go read the status word
```



```
                        *
                        * section for write
                        *
000003A0 08C0 000F f85_wrt bset    #15,d0                  make it a write command
000003A4 3940 0004         move    d0,4(a4)                place the command in the output buffer

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

000003B0 196E FFFF         move.b  v85gdcb(a6),3(a4)       set gpio card dma control bits
                 0003
000003B6 357C 0006         move    #$0006,6(a2)            arm the dma channel
                 0006

000003BC 46DF              move    (sp)+,sr                re-enable interrupts
                        ************************ interrupts re-enabled ************************
000003BE 082C 0003 f85_wwt btst    #3,7(a4)                peripheral status set?
                 0007
000003C4 660E              bne.s   f85_tdx                 if so, terminate dma xfer now!
000003C6 082A 0000         btst    #0,7(a2)                dma channel still armed?
                 0007
000003CC 66F0              bne.s   f85_wwt                 if so, loop

000003CE 6100 0036         bsr     f85cswf                 check status and wait for the flag
000003D2 4E71                nop                           ret 1; some error occurred

                        *
                        * terminate the dma transfer and read the status word
                        *
000003D4 3012           f85_tdx move    (a2),d0                 disarm the dma channel
000003D6 422C 0003              clr.b   3(a4)                   clr the gpio card dma control bits

000003DA 197C 0001              move.b  #1,7(a4)                set the transfer complete bit
                 0007
000003E0 6100 0050              bsr     f85winw                 word in; don't wait for the flag
000003E4 3D40 FFF8              move    d0,v85stat(a6)          save the status word for now
000003E8 197C 0000              move.b  #0,7(a4)                clear the transfer complete bit
                 0007

000003EE 7000                   moveq   #0,d0
000003F0 6100 0034              bsr     f85wo                   complete the handshake with the '85
                        *
                        * check for errors
                        *
000003F4 302E FFF8              move    v85stat(a6),d0          drive ready, seek complete,
000003F8 C07C FFF0              and     #$FFF0,d0                 transfer complete, &
000003FC B07C 0060              cmp     #$0060,d0                   no error in error code?
00000400 6602                   bne.s   *+4                     branch if any errors
00000402 5497                   addq.l  #2,(sp)                 setup a normal (ret 2) exit
00000404 4E75                   rts                             exit
```

```
                        ********************************************************************
                        *                        9885 shared driver routines               *
                        ********************************************************************
                        *
                        * routine to check status while waiting on the flag
                        *
00000406 5497           f85cswf addq.l  #2,(sp)                 anticipate a normal (ret 2) exit

00000408 0814 0000 f85tfb  btst    #0,(a4)                      flag bit
0000040C 660A              bne.s   f85ret                       branch if set

0000040E 082C 0003         btst    #3,7(a4)                     peripheral status bit
              0007
00000414 67F2              beq.s   f85tfb                       keep looping unless it's set

00000416 5597              subq.l  #2,(sp)                      change the ret 2 back to a ret 1
00000418 4E75           f85ret  rts                             exit


                        *
                        * routine to wait for the flag
                        *
0000041A 0814 0000 f85wf   btst    #0,(a4)                      flag bit
0000041E 67FA              beq     *-4                          loop until set
00000420 4E75              rts


                        *
                        * routine to send the 9885 password
                        *
00000422 303C AE87 f85login move    #44679,d0                   9885's "secret" password


                        *
                        * routine to send one word out on the gpio card
                        *
00000426 61F2           f85wo   bsr     f85wf                   wait for the flag
00000428 3940 0004              move    d0,4(a4)                place word in the output buffer
0000042C 1880                   move.b  d0,(a4)                 set the peripheral control line
0000042E 4E75                   rts
```



```
                        *
                        * routine to receive one word in on the gpio card
                        *
00000430 61E8           f85wi   bsr     f85wf                   wait for the flag

00000432 397C 0000 f85winw move    #0,4(a4)                     clear the output buffer
              0004

00000438 4A6C 0004              tst     4(a4)                   set direction in w/ dummy read
0000043C 1880                   move.b  d0,(a4)                 set the peripheral control line
0000043E 61DA                   bsr     f85wf                   wait for the flag
00000440 302C 0004              move    4(a4),d0                load in the word
00000444 4E75                   rts


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

FORMAT

Description

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                       *
    7                       *
    8                       * Last Updated August 19, 1983 - timing fixes
    9                       *
   10                       ************************************************
   11                       *
   12    00000000                    RORG    0
   13                                NOSYMS
   14
   15                                DEF     ASMR_FORMAT
   16                                REFA    SYSGLOBALS
   17                                REFA    XMINIT
   18                                REFA    CHECK_TIMER,DELAY_TIMER                 JS 8/19/83
   19                                LMODE   CHECK_TIMER,DELAY_TIMER                 JS 8/19/83
   20
   21              0000 0001 TIMER_PRESENT EQU 1                                     JS 8/19/83
   22              FFFF FEDA SYSFLAG2      EQU $FFFFFEDA                             JS 8/19/83
   23
   24
   25              0000 0010 PHI       EQU     $10
   26              0000 0001 CARD      EQU     $1
   27              0050 0000 DMAC      EQU     $500000  LOCATION OF BUILT IN DMA CARD
   28                        *
   29              FFFF FFEC INITISRIB EQU     XMINIT-20    INIT'S ISRIB
   30                        *
   31                        * PHI REGISTER ADDRESSES, IN ASCENDING ORDER
   32                        *
   33              0000 0011 INTR      EQU     PHI+1
   34              0000 0013 IMSK      EQU     PHI+3
   35              0000 0015 FIFOFFSET EQU     PHI+5
   36              0000 0017 STATPOFFS EQU     PHI+7
   37              0000 0019 CTRL      EQU     PHI+9
   38              0000 001B HPIBADR   EQU     PHI+$B
   39              0000 001D PPMSK     EQU     PHI+$D
   40              0000 001F PPSNS     EQU     PHI+$F
   41                        *
   42                        * PHI CARD REGISTERS
   43                        *
   44              0000 0003 CRDCTRL   EQU     CARD+2
   45                        *CRDSTAT  EQU     CARD+2
   46              0000 0001 ID        EQU     CARD
   47                        *
   48                        * DMA REGISTERS
   49                        *
   50              0050 0000 DMADR0    EQU     DMAC
   51              0050 0004 DMWCT0    EQU     DMAC+4
   52              0050 0006 DMCTRL0   EQU     DMAC+6
   53              0050 0000 DMSTAT    EQU     DMAC
   54              0050 0008 DMADR1    EQU     DMAC+8
   55              0050 000C DMWCT1    EQU     DMAC+$C
   56              0050 000E DMCTRL1   EQU     DMAC+$E
   57                        *
   58                        * AMIGO RELATED CONSTANTS
```



```
   59                        *
   60              0000 005E PTA       EQU     $5E         PHI TALK ADDRESS
   61              0000 003E PLA       EQU     $3E         PHI LISTEN ADDRESS
   62              0000 0040 DTAC      EQU     $40         DISK PRIMARY TALK BASE
   63              0000 0020 DLAC      EQU     $20         DISK PRIMARY LISTEN BASE
   64              0000 0070 DSJ       EQU     $70         DSJ SECONDARY
   65              0000 0068 DSAD      EQU     $68         DISK STANDARD SECONDARY
   66              0000 0040 ICMND     EQU     $40         INTERFACE COMMAND SETUP
   67              0000 0080 EOI       EQU     $80         EOI SETUP
   68              0000 005F UNT       EQU     $5F         UNTALK COMMAND
   69              0000 003F UNL       EQU     $3F         UNLISTEN COMMAND
   70                        *
   71                        *
   72              0000 0000 SECTOR    EQU     0
   73                        *
   74                        *
   75                        * REGISTER SETUP
   76                        *
   77              0000 0000 DTEMP   EQU     D0
   78              0000 0001 MTEMP   EQU     D1
   79              0000 0002 BCOUNT  EQU     D2
   80              0000 0003 TCOUNT  EQU     D3
   81              0000 0004 DZERO   EQU     D4
   82              0000 0005 FREE3   EQU     D5              (AVAILABLE FOR USE)
   83              0000 0006 FREE2   EQU     D6              (AVAILABLE FOR USE)
   84              0000 0007 FREE1   EQU     D7              (AVAILABLE FOR USE)
   85
   86              0000 0000 BYTEPTR EQU     A0
   87              0000 0001 FIFO    EQU     A1
   88              0000 0002 PORT    EQU     A2
   89              0000 0003 ATEMP   EQU     A3
   90              0000 0004 STATP   EQU     A4
   91              0000 0005 GLOBAL  EQU     A5
   92              0000 0006 BASE    EQU     A6
   93                        *SP     EQU     A7
   94
   95
   96                        *
   97                        *  PARAMETERS PASSED FROM PASCAL
   98                        *
   99
  100              0000 0020 PARAMS   EQU    40-8             NUMBER OF BYTES FOR PASSED PARAMETERS
  101              0000 0028 BUSADDR  EQU    38
  102              0000 0024 UNIT     EQU    36
  103              0000 0022 HEAD     EQU    34
  104              0000 001E PHIPORT  EQU    30
  105              0000 001A TRACKB   EQU    26
  106              0000 0016 SECTORS  EQU    22
  107              0000 0014 SPD      EQU    20
  108              0000 0010 MADR     EQU    16
  109              0000 000C TRACKA   EQU    12               ADDRESS OF TRACK A
  110              0000 0008 ERRTYPE  EQU     8
  111                        *RETADDR EQU     4               RETURN ADDRESS
  112                        *STATLNK EQU     0               STATIC LINK
  113                        *
  114                        *  LOCAL PARAMETERS
  115                        *
```

```
PAGE   3 [3.0] 12/26/84 22:58:02  ASSEMBLY OF FORMAT.TEXT                                   *** File name: FORMAT ***

 116                FFFF FFFF DLA      EQU    -1      DISC LISTEN ADDRESS
 117                FFFF FFFE DTA      EQU    -2      DISC TALK ADDRESS
 118                FFFF FFFD DSJBYTE  EQU    -3      DSJ BYTE
 119                FFFF FFFC STAT3    EQU    -4      STORAGE FOR STATUS BYTES
 120                FFFF FFFB STAT2    EQU    -5
 121                FFFF FFFA STAT1    EQU    -6
 122                FFFF FFF9 STAT0    EQU    -7
 123                FFFF FFF8 ADDR3    EQU    -8      STORAGE FOR DISC ADDRESS
 124                FFFF FFF7 ADDR2    EQU    -9
 125                FFFF FFF6 ADDR1    EQU    -10
 126                FFFF FFF5 ADDR0    EQU    -11
 127                FFFF FFF4 PMASK    EQU    -12     PPOLL MASK
 128                FFFF FFF3 IFLAG    EQU    -13     INTERRUPT FLAG
 129                FFFF FFF2 SPDTEMP  EQU    -14     UPPER 3 BITS=S,P,D
 130                FFFF FFF0 CYLINDR1 EQU    -16     CYL ADDR FOR SEEK
 131                FFFF FFEE CYLINDR2 EQU    -18     CYL ADDR FOR ADDR REC
 132                FFFF FFEA ADDRSTOR EQU    -22     STORAGE FOR SYSTEM INTERRUPT VECTOR
 133                FFFF FFE6 STATSTOR EQU    -26     STORAGE FOR SYSTEM STATIC LINK
 134                FFFF FFE6 LOCALS   EQU    -26     TOTAL AMOUNT OF LOCAL STOREAGE
 135                          *
 136                          *
 137    00000000 4E56 FFE6 ASMR_FORMAT  LINK    BASE,#LOCALS
 138    00000004 246E 001E         MOVEA.L PHIPORT(BASE),PORT      ADDRESS OF PHI CARD INTERFACE
 139    00000008 43EA 0015         LEA     FIFOFFSET(PORT),FIFO
 140    0000000C 49EA 0017         LEA     STATPOFFS(PORT),STATP
 141    00000010 4284              CLR.L   DZERO
 142
 143    00000012 47FA 041E         LEA     ISV,ATEMP               SET UP INTERRUPT VECTOR
 144    00000016 2B4B FFF4         MOVE.L  ATEMP,INITISRIB+8(GLOBAL)  ADDRESS OF ISR
 145    0000001A 2B4E FFF8         MOVE.L  BASE,INITISRIB+12(GLOBAL)  STATIC LINK
 146
 147    0000001E 122E 0027         MOVE.B  BUSADDR+1(BASE),MTEMP   CALCULATE PPOLL MASK
 148    00000022 103C 0080         MOVE.B  #$80,DTEMP
 149    00000026 E228              LSR.B   MTEMP,DTEMP
 150    00000028 1D40 FFF4         MOVE.B  DTEMP,PMASK(BASE)
 151
 152    0000002C 103C 0040         MOVE.B #DTAC,DTEMP              CALCULATE DISC TALK ADDRESS
 153    00000030 D02E 0027         ADD.B  BUSADDR+1(BASE),DTEMP
 154    00000034 1D40 FFFE         MOVE.B DTEMP,DTA(BASE)
 155
 156    00000038 103C 0020         MOVE.B #DLAC,DTEMP              CALCULATE DISC LISTEN ADDRESS
 157    0000003C D02E 0027         ADD.B  BUSADDR+1(BASE),DTEMP
 158    00000040 1D40 FFFF         MOVE.B DTEMP,DLA(BASE)
 159
 160    00000044 266E 0008         MOVEA.L ERRTYPE(BASE),ATEMP     LOAD ERRTYPE ADDRESS
 161    00000048 2684              MOVE.L  DZERO,(ATEMP)           CLEAR ERRTYPE
 162
 163    0000004A 6100 03B4         BSR  SFM     SET FILE MASK
 164
 165    0000004E 266E 000C         MOVEA.L TRACKA(BASE),ATEMP      COMPARE TRACK A & B
 166    00000052 2013              MOVE.L (ATEMP),DTEMP
 167    00000054 B0AE 001A         CMP.L  TRACKB(BASE),DTEMP
 168    00000058 6600 002C         BNE    SPARE
 169                          *    IF EQUAL THEN FORMAT TRACK A
 170    0000005C 6100 009A         BSR    DMSETUP
 171    00000060 266E 000C         MOVEA.L TRACKA(BASE),ATEMP
```

```
PAGE   4 [3.0] 12/26/84 22:58:02  ASSEMBLY OF FORMAT.TEXT                                   *** File name: FORMAT ***

 172    00000064 3D6B 0002         MOVE.W 2(ATEMP),CYLINDR1(BASE)
                 FFF0
 173    0000006A 302E 0014         MOVE.W SPD(BASE),DTEMP          SHIFT S,P,D TO UPPER 3 BITS
 174    0000006E C1FC 0020         MULS   #32,DTEMP
 175    00000072 1D40 FFF2         MOVE.B DTEMP,SPDTEMP(BASE)
 176
 177    00000076 6100 0238         BSR    SEEK
 178    0000007A 6100 0272         BSR    INITZE
 179    0000007E 6100 0092         BSR    ERCHK
 180    00000082 6000 006A         BRA    END_IO
 181
 182                          * NOW SPARE TRACK A WITH TRACK B
 183
 184    00000086 266E 000C SPARE   MOVEA.L TRACKA(BASE),ATEMP
 185    0000008A 3D6B 0002         MOVE.W 2(ATEMP),CYLINDR1(BASE)
                 FFF0
 186    00000090 3D6E 001C         MOVE.W TRACKB+2(BASE),CYLINDR2(BASE)
                 FFEE
 187    00000096 1D7C 0020         MOVE.B  #32,SPDTEMP(BASE)   SET D-BIT
                 FFF2
 188    0000009C 6100 0212         BSR    SEEK
 189    000000A0 6100 0306         BSR    ADDREC
 190    000000A4 6100 0052         BSR    DMSETUP
 191    000000A8 6100 0244         BSR    INITZE
 192    000000AC 6100 0064         BSR    ERCHK
 193    000000B0 102E FFFD         MOVE.B DSJBYTE(BASE),DTEMP
 194    000000B4 6600 0038         BNE    END_IO
 195    000000B8 102E FFF9         MOVE.B STAT0(BASE),DTEMP
 196    000000BC 0200 001F         ANDI.B #$1F,DTEMP   CLEAR SPD
 197    000000C0 6600 002C         BNE    END_IO
 198
 199    000000C4 3D6E 001C         MOVE.W TRACKB+2(BASE),CYLINDR1(BASE)
                 FFF0
 200    000000CA 266E 000C         MOVEA.L TRACKA(BASE),ATEMP
 201    000000CE 3D6B 0002         MOVE.W 2(ATEMP),CYLINDR2(BASE)
                 FFEE
 202    000000D4 1D7C 0080         MOVE.B  #128,SPDTEMP(BASE)  SET S-BIT
                 FFF2
 203    000000DA 6100 01D4         BSR    SEEK
 204    000000DE 6100 02C8         BSR    ADDREC
 205    000000E2 6100 0014         BSR    DMSETUP
 206    000000E6 6100 0206         BSR    INITZE
 207    000000EA 6100 0026         BSR    ERCHK
 208
 209    000000EE 4E5E      END_IO  UNLK   BASE
 210    000000F0 265F              MOVEA.L (SP)+,ATEMP
 211    000000F2 DEFC 0020         ADDA.W  #PARAMS,SP
 212    000000F6 4ED3              JMP     (ATEMP)
 213                          *
 214                          *
 215                          * DMA SETUP ROUTINE
 216                          * - AT THE END OF THIS ROUTINE THE DMA CARD WILL BE
 217                          *   SET UP FOR A DISK TRANSFER
 218                          *
 219    000000F8 23EE 0010 DMSETUP   MOVE.L   MADR(BASE),DMADR0
                 0050 0000
 220    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,DMWCTO    LOWER 2 BYTES ONLY
         0004
00000110 4E75               RTS
                    *
                    *
                    * Error Check Routine
                    *
00000112 6100 0042 ERCHK    BSR     DSJR
00000116 6100 007E          BSR     DSTAT
0000011A 102E FFFD          MOVE.B  DSJBYTE(BASE),DTEMP
0000011E 6600 000E          BNE     ERROR   CHECK DST BYTE
00000122 102E FFF9          MOVE.B  STAT0(BASE),DTEMP
00000126 0200 001F          ANDI.B  #$1F,DTEMP  CLEAR SPD BITS
0000012A 6700 0028          BEQ     FINISH  CHECK STATUS 1
0000012E 6100 0004 ERROR    BSR     RADDREC
00000132 206E 0008          MOVEA.L ERRTYPE(BASE),ATEMP
00000136 16EE FFF9          MOVE.B  STAT0(BASE),(ATEMP)+
0000013A 16EE FFFA          MOVE.B  STAT1(BASE),(ATEMP)+
0000013E 16EE FFFB          MOVE.B  STAT2(BASE),(ATEMP)+
00000142 16AE FFFC          MOVE.B  STAT3(BASE),(ATEMP)
00000146 206E 000C          MOVEA.L TRACKA(BASE),ATEMP
0000014A 36C4               MOVE.W  DZERO,(ATEMP)+
0000014C 16EE FFF5          MOVE.B  ADDR0(BASE),(ATEMP)+
00000150 16AE FFF6          MOVE.B  ADDR1(BASE),(ATEMP)
00000154 4E75       FINISH  RTS
                    *
                    *
                    *  DSJ ROUTINE
                    *
00000156 1884       DSJR    MOVE.B  DZERO,(STATP)   DISABLE INTERRUPTS
00000158 1544 0013          MOVE.B  DZERO,IMSK(PORT)
0000015C 18BC 0040          MOVE.B  #ICMND,(STATP)
00000160 12BC 003E          MOVE.B  #PLA,(FIFO)
00000164 12AE FFFE          MOVE.B  DTA(BASE),(FIFO)
00000168 12BC 0070          MOVE.B  #DSJ,(FIFO)
0000016C 143C 0001          MOVE.B  #$01,BCOUNT             LOAD INPUT COUNT
00000170 1884               MOVE.B  DZERO,(STATP)           INTERRUPT ENABLE
00000172 157C 0004          MOVE.B  #$04,IMSK(PORT)         ENABLE "FIFO BYTE"
         0013
00000178 41EE FFFD          LEA     DSJBYTE(BASE),BYTEPTR     STORE ADDR. FOR DSJ
0000017C 1884               MOVE.B  DZERO,(STATP)
0000017E 12BC 0001          MOVE.B  #$01,(FIFO)             ENABLE TRANSFER
00000182 4EBA 02D6          JSR     IWAIT                   WAIT FOR INTERRUPT
00000186 18BC 0040          MOVE.B  #ICMND,(STATP)
0000018A 12BC 005F          MOVE.B  #UNT,(FIFO)
0000018E 12BC 003F          MOVE.B  #UNL,(FIFO)
00000192 6000 010C ENDSJR   BRA     IDLE
                    *
                    *
                    *  REQUEST AND READ DISK STATUS
                    *
00000196 1544 0013 DSTAT    MOVE.B  DZERO,IMSK(PORT)
0000019A 18BC 0040          MOVE.B  #ICMND,(STATP)
0000019E 12BC 005E          MOVE.B  #PTA,(FIFO)
000001A2 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
```



```
000001A6 12BC 0068          MOVE.B  #DSAD,(FIFO)
000001AA 6100 00DC          BSR     PNWAIT
000001AE 1884               MOVE.B  DZERO,(STATP)
000001B0 12BC 0003          MOVE.B  #$03,(FIFO)     OP CODE
000001B4 18BC 0080          MOVE.B  #EOI,(STATP)
000001B8 12AE 0025          MOVE.B  UNIT+1(BASE),(FIFO)
000001BC 18BC 0040          MOVE.B  #ICMND,(STATP)
000001C0 12BC 003F          MOVE.B  #UNL,(FIFO)
000001C4 6100 00DA          BSR     IDLE
000001C8 18BC 0040          MOVE.B  #ICMND,(STATP)
000001CC 12BC 003E          MOVE.B  #PLA,(FIFO)
000001D0 12AE FFFE          MOVE.B  DTA(BASE),(FIFO)
000001D4 12BC 0068          MOVE.B  #DSAD,(FIFO)
000001D8 143C 0004          MOVE.B  #$04,BCOUNT             LOAD INPUT COUNT
000001DC 41EE FFF9          LEA     STAT0(BASE),BYTEPTR     STORE ADDR. FOR STAT
000001E0 1884               MOVE.B  DZERO,(STATP)
000001E2 157C 0004          MOVE.B  #$04,IMSK(PORT)     ENABLE "FIFO BYTE"
         0013
000001E8 18BC 0080          MOVE.B  #$80,(STATP)    LF INHIBIT
000001EC 12BC 0004          MOVE.B  #$04,(FIFO)     COUNTED XFER ENABLE
000001F0 4EBA 0268          JSR     IWAIT
000001F4 18BC 0040          MOVE.B  #ICMND,(STATP)
000001F8 12BC 005F          MOVE.B  #UNT,(FIFO)
000001FC 12BC 003F          MOVE.B  #UNL,(FIFO)
00000200 6000 009E          BRA     IDLE
                    *
                    *
                    *  REQUEST AND READ ADDRESS RECORD
                    *
00000204 1544 0013 RADDREC  MOVE.B  DZERO,IMSK(PORT)
00000208 18BC 0040          MOVE.B  #ICMND,(STATP)
0000020C 12BC 005E          MOVE.B  #PTA,(FIFO)
00000210 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
00000214 12BC 0068          MOVE.B  #DSAD,(FIFO)
00000218 6100 006E          BSR     PNWAIT
0000021C 1884               MOVE.B  DZERO,(STATP)
0000021E 12BC 0014          MOVE.B  #$14,(FIFO)     OP CODE
00000222 18BC 0080          MOVE.B  #EOI,(STATP)
00000226 12AE 0025          MOVE.B  UNIT+1(BASE),(FIFO)
0000022A 18BC 0040          MOVE.B  #ICMND,(STATP)
0000022E 12BC 003F          MOVE.B  #UNL,(FIFO)
00000232 6100 006C          BSR     IDLE
00000236 18BC 0040          MOVE.B  #ICMND,(STATP)
0000023A 12BC 003E          MOVE.B  #PLA,(FIFO)
0000023E 12AE FFFE          MOVE.B  DTA(BASE),(FIFO)
00000242 12BC 0068          MOVE.B  #DSAD,(FIFO)
00000246 143C 0004          MOVE.B  #$04,BCOUNT             LOAD INPUT COUNT
0000024A 41EE FFF5          LEA     ADDR0(BASE),BYTEPTR     STORE ADDR. FOR ADDRESS REC
0000024E 1884               MOVE.B  DZERO,(STATP)
00000250 157C 0004          MOVE.B  #$04,IMSK(PORT)     ENABLE "FIFO BYTE"
         0013
00000256 18BC 0080          MOVE.B  #$80,(STATP)    LF INHIBIT
0000025A 12BC 0004          MOVE.B  #$04,(FIFO)     COUNTED XFER ENABLE
0000025E 4EBA 01FA          JSR     IWAIT
00000262 18BC 0040          MOVE.B  #ICMND,(STATP)
00000266 12BC 005F          MOVE.B  #UNT,(FIFO)
0000026A 12BC 003F          MOVE.B  #UNL,(FIFO)
```

```
  331  0000026E 6000 0030          BRA     IDLE
  332                       *
  333                       *
  334                       * PPOLL HOLDOFF
  335                       *
  336  00000272 1544 001F PWAIT    MOVE.B  DZERO,PPSNS(PORT)             POSITIVE SENSE
  337  00000278 156E FFF4          MOVE.B  PMASK(BASE),PPMSK(PORT) SET PPMSK
                    001D
  338  0000027C 1884               MOVE.B  DZERO,(STATP)
  339  0000027E 157C 0020          MOVE.B  #$20,IMSK(PORT)         PPOLL RESPONSE
                    0013
  340  00000284 6000 01D4          BRA     IWAIT
  341                       *
  342                       *
  343                       *  PPOLL NEGATION HOLDOFF
  344                       *
  345  00000288 156E FFF4 PNWAIT   MOVE.B  PMASK(BASE),PPSNS(PORT) SET PPSNS
                    001F
  346  0000028E 156E FFF4          MOVE.B  PMASK(BASE),PPMSK(PORT) SET PPMSK
                    001D
  347  00000294 1884               MOVE.B  DZERO,(STATP)
  348  00000296 157C 0020          MOVE.B  #$20,IMSK(PORT)
                    0013
  349  0000029C 6000 01BC          BRA     IWAIT
  350                       *
  351                       *
  352                       *  FIFO IDLE HOLDOFF
  353                       *
  354  000002A0 1544 0013 IDLE     MOVE.B  DZERO,IMSK(PORT)
  355  000002A4 1884               MOVE.B  DZERO,(STATP)
  356  000002A6 157C 0002          MOVE.B  #$02,IMSK(PORT)    ENABLE "FIFO IDLE"
                    0013
  357  000002AC 6000 01AC          BRA     IWAIT
  358                       *
  359                       *
  360                       *  SEEK ROUTINE
  361                       *
  362  000002B0 18BC 0040 SEEK     MOVE.B  #ICMND,(STATP)
  363  000002B4 12BC 005E          MOVE.B  #PTA,(FIFO)
  364  000002B8 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
  365  000002BC 12BC 0068          MOVE.B  #DSAD,(FIFO)    SEEK SECONDARY
  366  000002C0 61C6               BSR     PNWAIT
  367  000002C2 1884               MOVE.B  DZERO,(STATP)
  368  000002C4 12BC 0002          MOVE.B  #$02,(FIFO)     OP CODE
  369  000002C8 12AE 0025          MOVE.B  UNIT+1(BASE),(FIFO)
  370  000002CC 12AE FFF0          MOVE.B  CYLINDR1(BASE),(FIFO)   CYLAD HIGH BYTE
  371  000002D0 12AE FFF1          MOVE.B  CYLINDR1+1(BASE),(FIFO)   CYLAD LOW BYTE
  372  000002D4 12AE 0023          MOVE.B  HEAD+1(BASE),(FIFO)
  373  000002D8 61C6               BSR     IDLE            EMPTY FIFO
  374  000002DA 18BC 0080          MOVE.B  #EOI,(STATP)
  375  000002DE 12BC 0000          MOVE.B  #SECTOR,(FIFO)
  376  000002E2 18BC 0040          MOVE.B  #ICMND,(STATP)
  377  000002E6 12BC 003F          MOVE.B  #UNL,(FIFO)
  378  000002EA 61B4               BSR     IDLE
  379  000002EC 6084               BRA     PWAIT     WAIT FOR SEEK COMPLETION
  380                       *
  381                       *
```



```
  382                       *  INITIALIZE
  383                       *
  384  000002EE 18BC 0040 INITZE   MOVE.B  #ICMND,(STATP)
  385  000002F2 12BC 005E          MOVE.B  #PTA,(FIFO)
  386  000002F6 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
  387  000002FA 12BC 0068          MOVE.B  #DSAD,(FIFO)
  388  000002FE 61B8               BSR     PNWAIT
  389  00000300 102E FFF2          MOVE.B  SPDTEMP(BASE),DTEMP
  390  00000304 0600 000B          ADDI.B  #$B,DTEMP         ADD SPD TO OP CODE
  391  00000308 1280               MOVE.B  DTEMP,(FIFO)
  392  0000030A 18BC 0080          MOVE.B  #EOI,(STATP)
  393  0000030E 12AE 0025          MOVE.B  UNIT+1(BASE),(FIFO)
  394  00000312 18BC 0040          MOVE.B  #ICMND,(STATP)
  395  00000316 12BC 003F          MOVE.B  #UNL,(FIFO)
  396  0000031A 6184               BSR     IDLE
  397
  398  0000031C 18BC 0040          MOVE.B  #ICMND,(STATP)
  399  00000320 12AE FFFF          MOVE.B  DLA(BASE),(FIFO)
  400  00000324 12BC 0060          MOVE.B  #$60,(FIFO)     SEC. WDAT
  401
  402  00000328 1884               MOVE.B  DZERO,(STATP)
  403  0000032A 1D44 FFF3          MOVE.B  DZERO,IFLAG(BASE)
  404  0000032E 33FC 000C          MOVE.W  #$0C,DMCTRL0    (PRIORITY,OUTPUT, BYTE)
                    0050 0006
  405  00000336 157C 0089          MOVE.B  #$89,CRDCTRL(PORT) (INTERRUPT,OUTPUT,CHAN. 0)
                    0003
  406  0000033C 157C 0002          MOVE.B  #$02,CTRL(PORT)   ENABLE DMARQ, 10 BIT PROC.?
                    0019
  407                       * NOW DATA IS BEING TRANSFERRED
  408                       *         MOVE.L  #363636,TCOUNT    2 SEC. TIMEOUT
  409  00000342 263C 0000          MOVE.L  #727,TCOUNT       1 MS AT 16 MHZ       JS 8/19/83
                    02D7
  410  00000348 4A2E FFF3 LOOPW    TST.B   IFLAG(BASE)
  411  0000034C 6620               BNE.S   WDONE
  412  0000034E 5383               SUBQ.L  #1,TCOUNT
  413  00000350 6CF6               BGE     LOOPW
  414  00000352 0838 0001          BTST    #TIMER_PRESENT,SYSFLAG2     CHECK FOR TIMER  JS 8/19/83
                    FEDA
  415  00000358 672A               BEQ.S   LOOPT                       USE IF PRESENT    JS 8/19/83
  416  0000035A 263C 0005          MOVE.L  #363636,TCOUNT        ELSE DO 2 SEC LOOP    JS 8/19/83
                    8C74
  417  00000360 4A2E FFF3 LOOPW2   TST.B   IFLAG(BASE)           SAME AS BEFORE        JS 8/19/83
  418  00000364 6608               BNE.S   WDONE                                     JS 8/19/83
  419  00000366 5383               SUBQ.L  #1,TCOUNT                                 JS 8/19/83
  420  00000368 6CF6               BGE     LOOPW2                                    JS 8/19/83
  421                       * FAILED TO DO IT IN 2 SECONDS
  422  0000036A 6000 0168          BRA     TESCAPE
  423  0000036E 157C 0080 WDONE    MOVE.B  #$80,CTRL(PORT)
                    0019
  424  00000374 18BC 0040          MOVE.B  #ICMND,(STATP)
  425  00000378 12BC 003F          MOVE.B  #UNL,(FIFO)
  426  0000037C 12BC 005F          MOVE.B  #UNT,(FIFO)
  427  00000380 6000 FF1E          BRA     IDLE
  428  00000384 1F3C 0001 LOOPT    MOVE.B  #1,-(SP)        SETUP TIMER ROUTINE RECORD    JS 8/19/83
  429  00000388 2F3C 0000          MOVE.L  #2000,-(SP)     FOR 2 SEC TIMEOUT             JS 8/19/83
                    07D0
  430  0000038E 4A2E FFF3 LOOPT2   TST.B   IFLAG(BASE)     SEE IF DONE                   JS 8/19/83
```

```
PAGE   9 [3.0] 12/26/84 22:58:02  ASSEMBLY OF FORMAT.TEXT                                   *** File name: FORMAT ***

  431    00000392 6610              BNE.S   LOOPT3          EXIT LOOP IF SO                 JS 8/19/83
  432    00000394 4857              PEA     (SP)            PUSH PTR TO RECORD              JS 8/19/83
  433    00000396 4EB9 0000         JSR     CHECK_TIMER     AND CHECK THE TIMER             JS 8/19/83
                  0000
  434    0000039C 6AF0              BPL     LOOPT2          LOOP IF NOT TIMED OUT           JS 8/19/83
  435    0000039E 5C4F              ADDQ    #6,SP           TIMEOUT, BUT GET A TRY          JS 5/2/84
  436    000003A0 760A              MOVEQ   #10,TCOUNT      ENTER NORMAL TIMEOUT LOOP       JS 5/2/84
  437    000003A2 60BC              BRA     LOOPW2          WITH A SHORT COUNT              JS 5/2/84
  438    000003A4 5C4F        LOOPT3 ADDQ   #6,SP           DONE, CLEANUP STACK             JS 8/19/83
  439    000003A6 60C6              BRA     WDONE                                           JS 8/19/83
  440                         *
  441                         *
  442                         *  ADDRESS RECORD ROUTINE
  443                         *
  444    000003A8 18BC 0040 ADDREC  MOVE.B  #ICMND,(STATP)
  445    000003AC 12BC 005E         MOVE.B  #PTA,(FIFO)
  446    000003B0 12AE FFFF         MOVE.B  DLA(BASE),(FIFO)
  447    000003B4 12BC 0068         MOVE.B  #DSAD,(FIFO)    ADDRESS RECORD SECONDARY
  448    000003B8 6100 FECE         BSR     PWAIT
  449    000003BC 1884              MOVE.B  DZERO,(STATP)
  450    000003BE 12BC 000C         MOVE.B  #$0C,(FIFO)     OP CODE
  451    000003C2 12AE 0025         MOVE.B  UNIT+1(BASE),(FIFO)
  452    000003C6 12AE FFEE         MOVE.B  CYLINDR2(BASE),(FIFO)   CYLAD HIGH BYTE
  453    000003CA 12AE FFEF         MOVE.B  CYLINDR2+1(BASE),(FIFO)   CYLAD LOW BYTE
  454    000003CE 12AE 0023         MOVE.B  HEAD+1(BASE),(FIFO)
  455    000003D2 6100 FECC         BSR     IDLE            EMPTY FIFO
  456    000003D6 18BC 0080         MOVE.B  #EOI,(STATP)
  457    000003DA 12BC 0000         MOVE.B  #SECTOR,(FIFO)
  458    000003DE 18BC 0040         MOVE.B  #ICMND,(STATP)
  459    000003E2 12BC 003F         MOVE.B  #UNL,(FIFO)
  460    000003E6 6000 FE8A         BRA     PWAIT     WAIT FOR COMPLETION
  461                         *
  462                         *
  463                         *  EMPTY INBOUND FIFO
  464                         *
  465    000003EA 1884        INEMP   MOVE.B  DZERO,(STATP)
  466    000003EC 157C 0004         MOVE.B  #$04,IMSK(PORT)
                  0013
  467    000003F2 4A2A 0011 EMPTY   TST.B   INTR(PORT)              CHECK FOR BYTE AVAILABLE
  468    000003F6 6700 0006         BEQ     EMPTY1                IF SO, READ, IF NOT RETURN.
  469    000003FA 4A11              TST.B   (FIFO)                (DUMMY READ OF FIFO TO EMPTY IT)
  470    000003FC 60F4              BRA     EMPTY
  471    000003FE 4E75        EMPTY1 RTS
  472                         *
  473                         *
  474                         *  SET FILE MASK ROUTINE
  475                         *     OUTPUT BYTE =XXXXDSCA
  476                         *     D=DECREMENTAL SEEK (1) OR INCREMENTAL SEEK(0)
  477                         *     S=1 ALLOWS AUTOMATIC SEEK TO SPARE TRACK
  478                         *     C=1 ENABLES CYLINDER MODE, C=0 SURFACE MODE
  479                         *     A=1 ENABLES AUTOMATIC SEEK AND END OF CYLINDER (SEE D)
  480                         *
  481                         *
  482    00000400 18BC 0040 SFM     MOVE.B  #ICMND,(STATP)
  483    00000404 12BC 005E         MOVE.B  #PTA,(FIFO)
  484    00000408 12AE FFFF         MOVE.B  DLA(BASE),(FIFO)
  485    0000040C 12BC 0068         MOVE.B  #DSAD,(FIFO)
```

```
PAGE  10 [3.0] 12/26/84 22:58:02  ASSEMBLY OF FORMAT.TEXT                                   *** File name: FORMAT ***

  486    00000410 6100 FE76         BSR     PWAIT
  487    00000414 1884              MOVE.B  DZERO,(STATP)
  488    00000416 12BC 000F         MOVE.B  #$0F,(FIFO)
  489    0000041A 18BC 0080         MOVE.B  #EOI,(STATP)
  490    0000041E 12BC 0001         MOVE.B  #1,(FIFO)
  491                         *   DISABLE AUTO SEEK TO SPARE,EN SURFACE, AUTO
  492                         *     SEEK TO NEXT CYLINDER
  493    00000422 18BC 0040         MOVE.B  #ICMND,(STATP)
  494    00000426 12BC 003F         MOVE.B  #UNL,(FIFO)
  495    0000042A 6100 FE74         BSR IDLE
  496    0000042E 6000 FE42         BRA PWAIT
  497                         *
  498                         *
  499                         *  INTERRUPT SERVICE ROUTINE
  500                         *
  501                         *
  502                         *  SERVICE EOP INTERRUPT
  503                         *
  504    00000432 265F        ISV     MOVEA.L (SP)+,ATEMP             RETURN ADDRESS
  505    00000434 2C5F              MOVEA.L (SP)+,BASE              STATIC LINK
  506    00000436 2E8B              MOVE.L  ATEMP,(SP)              (POP PARAMETER & PUSH) RETURN ADDRESS
  507    00000438 246E 001E         MOVEA.L PHIPORT(BASE),PORT
  508    0000043C 43EA 0015         LEA     FIFOFFSET(PORT),FIFO
  509    00000440 49EA 0017         LEA     STATPOFFS(PORT),STATP
  510    00000444 4284              CLR.L   DZERO
  511
  512    00000446 1544 0003         MOVE.B  DZERO,CRDCTRL(PORT)     DISABLE INTERRUPTS,DMA
  513    0000044A 4A2A 0019         TST.B   CTRL(PORT)              DUMMY READ TO CLEAR INTERRUPT
  514    0000044E 6100 FE22         BSR     PWAIT
  515    00000452 1D7C 0001         MOVE.B  #$01,IFLAG(BASE)
                  FFF3
  516    00000458 4E75              RTS
  517
  518                         *
  519                         *
  520                         *  IWAIT ROUTINE
  521                         *
  522    0000045A 363C 0258 IWAIT   MOVE.W   #600,TCOUNT       DO 1 MS AT 16 MHZ FAST LOOP    JS 8/19/83
  523    0000045E 102A 0011 IIWAIT  MOVE.B  INTR(PORT),DTEMP
  524    00000462 56CB FFFA         DBNE    TCOUNT,IIWAIT          TRY AGAIN
  525    00000466 661A              BNE.S   IWDONE              IF DONE THEN GET OUT OF HERE    JS 8/19/83
  526    00000468 0838 0001         BTST    #TIMER_PRESENT,SYSFLAG2   SEE IF WE HAVE TIMER    JS 8/19/83
                  FEDA
  527    0000046E 6726              BEQ.S   IWTIMER             USE IT IF WE DO                 JS 8/19/83
  528    00000470 263C 0000         MOVE.L  #36000,TCOUNT       ELSE USE 200 MS LOOP            JS 8/19/83
                  8CA0
  529    00000476 102A 0011 IIWAIT2 MOVE.B  INTR(PORT),DTEMP    CHECK FOR DONE                  JS 8/19/83
  530    0000047A 6606              BNE.S   IWDONE                                              JS 8/19/83
  531    0000047C 5383              SUBQ.L  #1,TCOUNT           THIS IS FIX FOR PREV BUG        JS 8/19/83
  532    0000047E 6EF6              BGT     IIWAIT2             CAN'T USE DBcc -- >32K COUNT    JS 8/19/83
  533    00000480 6052              BRA.S   TESCAPE             TIMED OUT                       JS 8/19/83
  534
  535    00000482 B03C 0004 IWDONE  CMP.B   #04,DTEMP
  536    00000486 6700 0032         BEQ     BYTE                    CHECK INTERRUPT TYPE
  537    0000048A B03C 0020         CMP.B   #$20,DTEMP
  538    0000048E 6700 0042         BEQ     PPW
  539    00000492 6000 0038         BRA     OUT
```

```
PAGE  11 [3.0] 12/26/84 22:58:02  ASSEMBLY OF FORMAT.TEXT                                   *** File name: FORMAT ***

  540                           *
  541     00000496 1F3C 0001 IWTIMER MOVE.B  #1,-(SP)          SETUP TIMER RECORD            JS 8/19/83
  542     0000049A 2F3C 0000         MOVE.L  #200,-(SP)        FOR 200 MS TIMEOUT            JS 8/19/83
                   00C8
  543     000004A0 102A 0011 IWTLOOP MOVE.B  INTR(PORT),DTEMP  CHECK FOR DONE                JS 8/19/83
  544     000004A4 6610              BNE.S   IWTEXIT           IF SO THEN GET OUT OF LOOP    JS 8/19/83
  545     000004A6 4857              PEA     (SP)              ELSE PUSH PTR TO TIME REC     JS 8/19/83
  546     000004A8 4EB9 0000         JSR     CHECK_TIMER       AND CHECK THE TIMER           JS 8/19/83
                   0000
  547     000004AE 6AF0              BPL     IWTLOOP           LOOP IF NOT TIMED OUT         JS 8/19/83
  548     000004B0 5C4F              ADDQ    #6,SP             TIMED OUT, TAKE ANOTHER TRY   JS 5/2/84
  549     000004B2 760A              MOVEQ   #10,TCOUNT        USE A SHORT COUNTER           JS 5/2/84
  550     000004B4 60C0              BRA     IIWAIT2           AND ENTER OTHER LOOP          JS 5/2/84
  551     000004B6 5C4F      IWTEXIT ADDQ    #6,SP             CLEANUP STACK WHEN DONE       JS 8/19/83
  552     000004B8 60C8              BRA     IWDONE            AND DO FINISH PROCESSING      JS 8/19/83
  553
  554
  555                           *  SERVICE FIFO BYTE AVAILABLE INTERRUPT
  556                           *
  557     000004BA 10D1      BYTE    MOVE.B  (FIFO),(BYTEPTR)+       STORE INPUT DATA
  558     000004BC 5302              SUBQ.B  #1,BCOUNT                   DECREMENT INPUT COUNT
  559     000004BE 6700 000C         BEQ     OUT
  560     000004C2 0C2A 0004 TEST    CMPI.B  #4,INTR(PORT)           ANOTHER BYTE?
                   0011
  561     000004C8 66F8              BNE     TEST
  562     000004CA 60EE              BRA     BYTE
  563     000004CC 1884      OUT     MOVE.B  DZERO,(STATP)           DISABLE INTERRUPTS
  564     000004CE 1544 0013         MOVE.B  DZERO,IMSK(PORT)
  565                           *
  566                           *
  567                           *  SERVICE PPOLL INTERRUPT
  568                           *
  569     000004D2 4E75      PPW     RTS
  570
  571     000004D4 6100 000E TESCAPE BSR     HDWRCLEAR          INVOKE IFC, CLEAR DMA, ETC.
  572     000004D8 3B7C FFF6         MOVE.W  #-10,SYSGLOBALS-2(GLOBAL)         ESCAPE(-10)
                   FFFE
  573     000004DE 2E6D FFF6         MOVEA.L SYSGLOBALS-10(GLOBAL),SP
  574     000004E2 4E75              RTS
  575                           *
  576                           *
  577                           * HARDWARE CLEAR
  578                           *
  579              0000 04E4 HDWRCLEAR EQU *
  580     000004E4 4A79 0050         TST.W   DMSTAT                  READ DMA CARD CONTROL REG TO DISARM IT
                   0000
  581     000004EA 1544 0001         MOVE.B  DZERO,ID(PORT)             SOFTWARE RESET
  582     000004EE 157C 0080         MOVE.B  #$80,HPIBADR(PORT)      ON-LINE
                   001B
  583     000004F4 6100 0010         BSR     IFCLEAR                 ASSERT IFC
  584                           * DISK CLEAR FOR HANGUPS
  585     000004F8 157C 0040         MOVE.B  #ICMND,STATPOFFS(PORT)
                   0017
  586     000004FE 157C 0014         MOVE.B  #$14,FIFOFFSET(PORT)    DEVICE CLEAR
                   0015
  587     00000504 4E75              RTS
  588                           *
```

```
PAGE  12 [3.0] 12/26/84 22:58:02  ASSEMBLY OF FORMAT.TEXT                                   *** File name: FORMAT ***

  589                           *
  590                           * INTERFACE CLEAR
  591                           *
  592     00000506 157C 0011 IFCLEAR MOVE.B  #$11,CTRL(PORT)         IFC, INIT OUTBOUND FIFO
                   0019
  593     0000050C 2F3C 0000 DEL100  MOVE.L  #100,-(SP)             100 MICROSECOND DELAY JS 8/19/83
                   0064
  594     00000512 4EB9 0000         JSR     DELAY_TIMER                                   JS 8/19/83
                   0000
  595     00000518 157C 0080         MOVE.B  #$80,CTRL(PORT)         8-BIT PROCESSOR
                   0019
  596     0000051E 4E75              RTS
  597                           *
  598                           *
  599                           *
  600                             END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

GASSM

Description

GASSM provides the low-level alpha driver for the high-resolution (1024×768) bit-mapped display.

Usage

Used only with the 9837A display.

Notes

This module is linked with the module GCRT (Pascal listings) to produce the INITLIB module CRTB.

```
PAGE   1 [3.0] 12/26/84 21:09:10  ASSEMBLY OF GASSM.TEXT                                   *** File name: GASSM ***

PASS 1 COMPLETE. ERRORS: 0
    1                           *
    2                           *   GATOR bit-mapped alpha driver
    3                           *
    4                           *     Pascal 3.0 version
    5                           *
    6                                   def     cscrollup,cscrolldown,cupdatecursor,cchar,cclear
    7                                   def     cbuildtable,cshiftleft,cshiftright
    8                                   def     cexchange,cscrollwindow,changecursor
    9                                   def     cscrollwinddn,cdbscroll1,cdbscrollr,cdbhighl
   10    00000000                       rorg.l 0
   11                                   refa    crtb,sysdevs
   12                                   nosyms
   13
   14              000C C000 clearl    equ     $CC000      blank pixel row offset
   15
   16              FFFF FFF6 maxx      equ     crtb-10
   17              FFFF FFF4 maxy      equ     crtb-12
   18              FFFF FFFC cursoraddr equ    crtb-4
   19              FFFF FFEE highlight equ     crtb-18
   20              FFFF FFAA controladdr equ   sysdevs-86
   21              FFFF FFA6 screen    equ     sysdevs-90
   22              FFFF FFA4 replcopy equ      sysdevs-92
   23              FFFF FFA2 windcopy equ      sysdevs-94
   24
   25              0000 4008 replreg   equ     $4008
   26              0000 400C windreg   equ     $400c
   27              0000 4001 status    equ     $4001
   28
   29              0000 0400 width         equ     1024
   30              0000 0023 initoffset    equ      $23                    offset to initialization offset
   31              0000 003B fontoffset    equ      $3B                    offset to font info offset
   32
   33                           *   gbuildtable(ptr);
   34
   35    00000000 285F       cbuildtable movea.l (sp)+,a4  a4 = return address
   36    00000002 206D FFAA              movea.l controladdr(a5),a0    get pointer to ROM start
   37    00000006 7000                   moveq   #0,d0
   38    00000008 7200                   moveq   #0,d1
   39    0000000A 1028 4001              move.b  status(a0),d0          get status reg again
   40    0000000E E408                   lsr.b   #2,d0                 get monitor type bits
   41    00000010 C03C 000C              and.b   #12,d0
   42    00000014 1230 0023              move.b  initoffset(a0,d0.w),d1  get MSB of info addr offset
   43    00000018 E149                   lsl.w   #8,d1
   44    0000001A 1230 0025              move.b  initoffset+2(a0,d0.w),d1  get LSB of info addr offset
   45    0000001E 2248                   movea.l a0,a1                 make copy of ROM start addr
   46    00000020 D2C1                   adda    d1,a1                 a1 points to init info now
   47    00000022 7200       ginitblock  moveq   #0,d1                 clear some regs
   48    00000024 7000                   moveq   #0,d0
   49    00000026 1029 0002              move.b  2(a1),d0              get word count to initialize
   50    0000002A 0309 0004              movep   4(a1),d1              form destination offset
   51    0000002E D288                   add.l   a0,d1                 d1 points to dest addr
   52    00000030 45E9 0008              lea     8(a1),a2              a2 points to first data byte
   53    00000034 2641                   movea.l d1,a3                 a3 points to destination
   54    00000036 030A 0000 ginitloop    movep   0(a2),d1               form a data word in d1
   55    0000003A 36C1                   move.w  d1,(a3)+              move data to the destination addr
   56    0000003C 0811 0006              btst    #6,(a1)               increment data pointer
   57    00000040 6602                   bne.s   ginit1                based on control byte
   58    00000042 584A                   addq    #4,a2
```

```
PAGE   2 [3.0] 12/26/84 21:09:10  ASSEMBLY OF GASSM.TEXT                                   *** File name: GASSM ***

   59    00000044 51C8 FFF0 ginit1      dbra    d0,ginitloop          loop till word count exhausted
   60    00000048 0811 0007             btst    #7,(a1)               was this last block?
   61    0000004C 660C                  bne.s   ginitdone             yes -- go return
   62    0000004E 0811 0006             btst    #6,(a1)               adjust data pointer
   63    00000052 6702                  beq.s   ginit2                to point to next init block
   64    00000054 584A                  addq    #4,a2
   65    00000056 224A       ginit2     movea.l a2,a1                 a1 points to new init block
   66    00000058 60C8                  bra     ginitblock            do the initialize
   67    0000005A 317C 0080 ginitdone   move.w  #128,replreg(a0)  set repl rule to clear
              4008
   68    00000060 317C 0000             move.w  #0,windreg(a0)
              400C
   69    00000066 7000                  moveq   #0,d0
   70    00000068 1028 4001             move.b  status(a0),d0
   71    0000006C C07C 000F             and     #15,d0                get frame buffer location
   72    00000070 7214                  moveq   #20,d1
   73    00000072 E3A8                  lsl.l   d1,d0                 put it in right place
   74    00000074 2B40 FFA6             move.l  d0,screen(a5)
   75    00000078 2240                  movea.l d0,a1                 clear the whole frame buffer
   76    0000007A 303C 03FB             move    #1019,d0               except last 4 pixel lines
   77    0000007E 12BC 0000 zloop       move.b  #00,(a1)
   78    00000082 D3FC 0000             adda.l  #width,a1
              0400
   79    00000088 0828 0007 zcheck      btst    #7,status(a0)
              4001
   80    0000008E 67F8                  beq     zcheck
   81    00000090 51C8 FFEC             dbra    d0,zloop
   82    00000094 317C 0003             move    #3,replreg(a0)
              4008
   83    0000009A 3B7C 0003             move    #3,replcopy(a5)
              FFA4
   84    000000A0 426D FFA2             clr     windcopy(a5)
   85    000000A4 0308 003B             movep   fontoffset(a0),d1     get font info offset
   86    000000A8 43F0 1002             lea     2(a0,d1.w),a1         point to font id code
   87    000000AC 7E02                  moveq   #2,d7                 count number of font found with d7
   88    000000AE 0509 0002 fontidchk   movep   2(a1),d2              get offset of font info
   89    000000B2 47F0 200A             lea     10(a0,d2.w),a3        a3 points to first char of font
   90    000000B6 0C11 0001             cmpi.b  #1,(a1)               is font = roman8 ?
   91    000000BA 6718                  beq.s   unpkroman             if so go unpack it
   92    000000BC 0C11 0002             cmpi.b  #2,(a1)               is font = kana8 upper half?
   93    000000C0 6738                  beq.s   unpkkana              if so go unpack it
   94    000000C2 5C49       nextfont   addq    #6,a1                 point to next font id
   95    000000C4 4A47                  tst     d7                    have we found both fonts?
   96    000000C6 66E6                  bne     fontidchk             if not look at this one
   97    000000C8 2B6D FFA6             move.l  screen(a5),cursoraddr(a5)  initialize cursor location
              FFFC
   98    000000CE 6100 02CC             bsr     changecursor          turn it on
   99    000000D2 4ED4                  jmp     (a4)                  return
  100
  101    000000D4 363C 0100 unpkroman   move    #256,d3               #chars to unpack
  102    000000D8 247C 000C             movea.l #$C0000,a2            start at beginning of font storage
              0000
  103    000000DE D5ED FFA6 unpackit    adda.l  screen(a5),a2
  104    000000E2 5347                  subq    #1,d7                 count a found font
  105    000000E4 E94B                  lsl     #4,d3                 get number of pixel rows to unpack
  106    000000E6 5343                  subq    #1,d3
  107    000000E8 7807       unpackrow  moveq   #7,d4                 we need to look at 8 bits/byte
```

```
PAGE   3 [3.0] 12/26/84 21:09:10  ASSEMBLY OF GASSM.TEXT                                   *** File name: GASSM ***

  108    000000EA 0913      unpackrow2  btst    d4,(a3)              is bit set in font?
  109    000000EC 56DA                  sne     (a2)+                set frame buffer byte accordingly
  110    000000EE 51CC FFFA             dbra    d4,unpackrow2        loop till all 8 bits done
  111    000000F2 544B                  addq    #2,a3                look at next font byte
  112    000000F4 51CB FFF2             dbra    d3,unpackrow         and loop till all font rows done
  113    000000F8 60C8                  bra     nextfont             go look at next font
  114    000000FA 363C 0080 unpkkana    move    #128,d3              kana8 upper half has 128 chars
  115    000000FE 247C 000C             movea.l #$C8000,a2           store at font storage + 256*128
                  8000
  116    00000104 60D8                  bra     unpackit
  117
  118
  119
  120                       * savecrtstate: preserve bit mover  state
  121                       *               Entry: d0= replacement rule
  122                       *                      d1= window width
  123                       *
  124                       *               Uses:  a2,a3
  125                       *
  126             0000 0106 savecrtstate equ *
  127    00000106 266D FFAA             movea.l controladdr(a5),a3
  128    0000010A 082B 0007 savestate1  btst   #7,status(a3)    wait for not busy
                  4001
  129    00000110 67F8                  beq    savestate1
  130    00000112 245F                  movea.l (sp)+,a2       save ret addr
  131    00000114 3F2D FFA4             move   replcopy(a5),-(sp)   save old copy
  132    00000118 3F2D FFA2             move   windcopy(a5),-(sp)
  133    0000011C 3B40 FFA4             move   d0,replcopy(a5)      setup new values
  134    00000120 3B41 FFA2             move   d1,windcopy(a5)
  135    00000124 3740 4008             move   d0,replreg(a3)           setup the registers
  136    00000128 3741 400C             move   d1,windreg(a3)
  137
  138    0000012C 4ED2                  jmp    (a2)
  139                       *
  140                       * restcrtstate: restores window width and replacement rule regs
  141                       *
  142                       *               Uses:  a3
  143                       *
  144             0000 012E restcrtstate equ *
  145    0000012E 266D FFAA             movea.l controladdr(a5),a3
  146    00000132 082B 0007 restcrt1    btst   #7,status(a3)      wait for not busy
                  4001
  147    00000138 67F8                  beq    restcrt1
  148    0000013A 3B6F 0004             move   4(sp),windcopy(a5)  restore copy variables
                  FFA2
  149    00000140 3B6F 0006             move   6(sp),replcopy(a5)
                  FFA4
  150    00000146 376D FFA2             move   windcopy(a5),windreg(a3)  restore the registers
                  400C
  151    0000014C 376D FFA4             move   replcopy(a5),replreg(a3)
                  4008
  152    00000152 2E9F                  move.l (sp)+,(sp)            move up return addr
  153    00000154 4E75                  rts                          and return
  154
  155                       *   procedure cchar(ord(char),x,y:shortint);
  156
  157    00000156 285F      cchar       movea.l (sp)+,a4
```

```
PAGE   4 [3.0] 12/26/84 21:09:10  ASSEMBLY OF GASSM.TEXT                                   *** File name: GASSM ***

  158    00000158 301F                  move    (sp)+,d0  d0 = y
  159    0000015A C0FC 4000             mulu    #16384,d0
  160    0000015E 2040                  movea.l d0,a0
  161    00000160 D1ED FFA6             adda.l  screen(a5),a0
  162    00000164 3A1F                  move    (sp)+,d5  d5 = x (this will be used later also)
  163    00000166 E74D                  lsl     #3,d5
  164    00000168 D0C5                  adda    d5,a0     a0 = address of byte to begin at
  165    0000016A 226D FFA6             movea.l screen(a5),a1    setup font addr in a1
  166    0000016E D3FC 000C             adda.l  #$C0000,a1       fonts are just past visible space
                  0000
  167    00000174 301F                  move    (sp)+,d0  d0 = character
  168    00000176 C0FC 0080             mulu    #128,d0
  169    0000017A 43F1 0800             lea     0(a1,d0.l),a1  a1 = address of char in font storage
  170    0000017E 3E3C 03F8             move    #width-8,d7
  171    00000182 303C 0003             move    #3,d0       set repl rule to replace
  172    00000186 082D 0000             btst    #0,highlight(a5)   inverse video?
                  FFEE
  173    0000018C 6704                  beq.s   ccharb              if not, skip next instruction
  174    0000018E 303C 000C             move    #12,d0              else set repl rule to invert
  175    00000192 7200      ccharb      moveq   #0,d1
  176    00000194 6100 FF70             bsr     savecrtstate
  177    00000198 20D9                  move.l  (a1)+,(a0)+
  178    0000019A 20D9                  move.l  (a1)+,(a0)+
  179    0000019C D0C7                  adda    d7,a0
  180    0000019E 20D9                  move.l  (a1)+,(a0)+
  181    000001A0 20D9                  move.l  (a1)+,(a0)+
  182    000001A2 D0C7                  adda    d7,a0
  183    000001A4 20D9                  move.l  (a1)+,(a0)+
  184    000001A6 20D9                  move.l  (a1)+,(a0)+
  185    000001A8 D0C7                  adda    d7,a0
  186    000001AA 20D9                  move.l  (a1)+,(a0)+
  187    000001AC 20D9                  move.l  (a1)+,(a0)+
  188    000001AE D0C7                  adda    d7,a0
  189    000001B0 20D9                  move.l  (a1)+,(a0)+
  190    000001B2 20D9                  move.l  (a1)+,(a0)+
  191    000001B4 D0C7                  adda    d7,a0
  192    000001B6 20D9                  move.l  (a1)+,(a0)+
  193    000001B8 20D9                  move.l  (a1)+,(a0)+
  194    000001BA D0C7                  adda    d7,a0
  195    000001BC 20D9                  move.l  (a1)+,(a0)+
  196    000001BE 20D9                  move.l  (a1)+,(a0)+
  197    000001C0 D0C7                  adda    d7,a0
  198    000001C2 20D9                  move.l  (a1)+,(a0)+
  199    000001C4 20D9                  move.l  (a1)+,(a0)+
  200    000001C6 D0C7                  adda    d7,a0
  201    000001C8 20D9                  move.l  (a1)+,(a0)+
  202    000001CA 20D9                  move.l  (a1)+,(a0)+
  203    000001CC D0C7                  adda    d7,a0
  204    000001CE 20D9                  move.l  (a1)+,(a0)+
  205    000001D0 20D9                  move.l  (a1)+,(a0)+
  206    000001D2 D0C7                  adda    d7,a0
  207    000001D4 20D9                  move.l  (a1)+,(a0)+
  208    000001D6 20D9                  move.l  (a1)+,(a0)+
  209    000001D8 D0C7                  adda    d7,a0
  210    000001DA 20D9                  move.l  (a1)+,(a0)+
  211    000001DC 20D9                  move.l  (a1)+,(a0)+
  212    000001DE D0C7                  adda    d7,a0
```

```
000001E0 20D9                move.l  (a1)+,(a0)+
000001E2 20D9                move.l  (a1)+,(a0)+
000001E4 D0C7                adda    d7,a0
000001E6 20D9                move.l  (a1)+,(a0)+
000001E8 20D9                move.l  (a1)+,(a0)+
000001EA D0C7                adda    d7,a0
000001EC 20D9                move.l  (a1)+,(a0)+
000001EE 20D9                move.l  (a1)+,(a0)+
000001F0 D0C7                adda    d7,a0
000001F2 20D9                move.l  (a1)+,(a0)+
000001F4 20D9                move.l  (a1)+,(a0)+
000001F6 082D 0002           btst    #2,highlight(a5)     underline?
         FFEE
000001FC 6716                beq.s   cchar1               no, skip next part
000001FE 6100 FF2E           bsr     restcrtstate
00000202 303C 008A           move    #138,d0              setup to invert line 16
00000206 72F8                moveq   #-8,d1
00000208 6100 FEFC           bsr     savecrtstate
0000020C D0FC FFF8           adda    #-8,a0               a0 points to line 16
00000210 108C 0000           move.b  #0,(a0)              do the invert
00000214 6100 FF18 cchar1    bsr     restcrtstate
00000218 4ED4                jmp     (a4)
                        *
                        *
                        *   cscrollup;
                        *
                        *   scrolls the screen up one line of alpha text (16 graphics lines)
                        *
0000021A 6100 0180 cscrollup bsr     changecursor
0000021E 206D FFA6           movea.l screen(a5),a0
00000222 2248                movea.l a0,a1
00000224 D0FC 4000           adda    #16384,a0
00000228 303C 0083           move.w  #131,d0
0000022C 7200                moveq   #0,d1
0000022E 6100 FED6           bsr     savecrtstate
00000232 302D FFF4           move    maxy(a5),d0
00000236 5340                subq    #1,d0
00000238 720F        suloop2 moveq   #15,d1
0000023A 1290        suloop  move.b  (a0),(a1)
0000023C D0FC 0400           adda    #width,a0
00000240 D2FC 0400           adda    #width,a1
00000244 082B 0007 sucheck   btst    #7,status(a3)   a3 setup by savecrtstate
         4001
0000024A 67F8                beq     sucheck
0000024C 51C9 FFEC           dbra    d1,suloop
00000250 51C8 FFE6           dbra    d0,suloop2

                        * Clear bottom line on screen
                        *
00000254 207C 000C           movea.l #clear1,a0
         C000
0000025A D1ED FFA6           adda.l  screen(a5),a0
0000025E 303C 000F           move    #15,d0
00000262 1290        bclrloop move.b  (a0),(a1)
00000264 D2FC 0400           adda    #width,a1
00000268 082B 0007 bcheck    btst    #7,status(a3)
         4001
```



```
0000026E 67F8                beq     bcheck
00000270 51C8 FFF0           dbra    d0,bclrloop
00000274 6100 FEB8           bsr     restcrtstate

00000278 6000 0122           bra     changecursor

                        *
                        *    cscrolldown
                        *
                        *    scrolls the screen down one text line
                        *
0000027C 6100 011E cscrolldown bsr     changecursor
00000280 206D FFA6           movea.l screen(a5),a0
00000284 302D FFF4           move    maxy(a5),d0
00000288 C0FC 4000           mulu    #16384,d0
0000028C 90BC 0000           sub.l   #width,d0
         0400
00000292 D1C0                adda.l  d0,a0
00000294 2248                movea.l a0,a1
00000296 D0FC 4000           adda    #16384,a0            point to 1 char row past a1
0000029A 303C 0083           move.w  #131,d0
0000029E 7200                moveq   #0,d1
000002A0 6100 FE64           bsr     savecrtstate
000002A4 302D FFF4           move    maxy(a5),d0
000002A8 5340                subq    #1,d0
000002AA 720F        sdloop2 moveq   #15,d1
000002AC 1091        sdloop  move.b  (a1),(a0)
000002AE 90FC 0400           suba    #width,a0
000002B2 92FC 0400           suba    #width,a1
000002B6 082B 0007 sdcheck   btst    #7,status(a3)    a3 setup by savecrtstate
         4001
000002BC 67F8                beq     sdcheck
000002BE 51C9 FFEC           dbra    d1,sdloop
000002C2 51C8 FFE6           dbra    d0,sdloop2

000002C6 206D FFA6           movea.l screen(a5),a0
000002CA 700F                moveq   #15,d0
000002CC 227C 000C           movea.l #clear1,a1
         C000
000002D2 D3C8                adda.l  a0,a1

000002D4 1091        topclear move.b  (a1),(a0)
000002D6 D0FC 0400           adda    #width,a0
000002DA 082B 0007 topcheck  btst    #7,status(a3)    a3 setup by savecrtstate
         4001
000002E0 67F8                beq     topcheck
000002E2 51C8 FFF0           dbra    d0,topclear
000002E6 6100 FE46           bsr     restcrtstate
000002EA 6000 00B0           bra     changecursor

                        *
                        * cupdatecursor(x,y:shortint);
                        *

000002EE 226D FFFC cupdatecursor movea.l cursoraddr(a5),a1
000002F2 285F                movea.l (sp)+,a4     a4 = return addr
000002F4 3A1F                move    (sp)+,d5     d5 = y
```

```
  319    000002F6 361F                    move    (sp)+,d3      d3 = x
  320    000002F8 303C 008A               move.w  #138,d0
  321    000002FC 72F8                    moveq   #-8,d1
  322    000002FE 6100 FE06               bsr     savecrtstate
  323    00000302 12BC 0000               move.b  #0,(a1)
  324    00000306 D2FC 0400               adda    #width,a1
  325    0000030A 082B 0007 curscheck     btst    #7,status(a3)  a3 setup in savecrtstate
                4001
  326    00000310 67F8                    beq     curscheck
  327    00000312 12BC 0000               move.b  #0,(a1)
  328
  329    00000316 CAFC 4000               mulu    #16384,d5     16*1024
  330    0000031A DABC 0000               add.l   #14336,d5   spaces you to line 15 of character for cursor
                3800
  331    00000320 E74B                    lsl     #3,d3
  332    00000322 206D FFA6               movea.l screen(a5),a0
  333    00000326 D0C3                    adda    d3,a0
  334    00000328 D1C5                    adda.l  d5,a0
  335    0000032A 2B48 FFFC               move.l  a0,cursoraddr(a5)
  336    0000032E 082B 0007 curscheck1    btst    #7,status(a3)
                4001
  337    00000334 67F8                    beq     curscheck1
  338    00000336 10BC 0000               move.b  #0,(a0)
  339    0000033A D0FC 0400               adda    #width,a0
  340    0000033E 082B 0007 curcheck2     btst    #7,status(a3)
                4001
  341    00000344 67F8                    beq     curcheck2
  342    00000346 10BC 0000               move.b  #0,(a0)
  343    0000034A 6100 FDE2 curcheck3     bsr     restcrtstate
  344
  345    0000034E 4ED4                    jmp     (a4)
  346
  347                     *
  348                     *  cclear(xpos,ypos,nchars:shortint);  REVISED FOR 3.01 9/13/84
  349                     *     -- clears nchars starting at xpos, ypos
  350                     *     -- nchars + xpos must not exceed 128
  351                     *        no range checking is done
  352                     *
  353
  354    00000350 6100 004A cclear        bsr     changecursor
  355    00000354 285F                    movea.l (sp)+,a4      a4 = return address
  356    00000356 381F                    move    (sp)+,d4      d4 = number of characters to clear
  357    00000358 361F                    move    (sp)+,d3      d3 = y to begin at
  358    0000035A C6FC 4000               mulu    #16384,d3     d3.l = offset to y
  359    0000035E 3A1F                    move    (sp)+,d5      d5 = x
  360    00000360 2F0C                    move.l  a4,-(sp)      stack return address
  361    00000362 E74D                    lsl     #3,d5         d5 = byte offset to begin at
  362    00000364 206D FFA6               movea.l screen(a5),a0
  363    00000368 227C 000C               movea.l #clear1,a1
                C000
  364    0000036E D3C8                    adda.l  a0,a1         blank line addr in a1
  365    00000370 D1C3                    adda.l  d3,a0         a0 = where to begin it all
  366    00000372 D0C5                    adda    d5,a0         after adding x offset
  367
  368    00000374 3604                    move    d4,d3         use requested length
  369    00000376 E74B                    lsl     #3,d3         convert to pixels
  370    00000378 4443                    neg     d3            complement
```



```
  371    0000037A 3203                    move.w  d3,d1
  372    0000037C 303C 0083               move.w  #131,d0
  373    00000380 6100 FD84               bsr     savecrtstate setup control regs
  374
  375    00000384 760F                    moveq   #15,d3        16 pixel rows per character line
  376    00000386 1091          clearpart move.b  (a1),(a0)
  377    00000388 D0FC 0400               adda    #width,a0
  378    0000038C 082B 0007 clearcheck    btst    #7,status(a3)  a3 setup in savecrtstate
                4001
  379    00000392 67F8                    beq     clearcheck
  380    00000394 51CB FFF0               dbra    d3,clearpart
  381    00000398 6100 FD94               bsr     restcrtstate
  382              0000 039C doneclear     equ     *
  383
  384
  385    0000039C 226D FFFC changecursor  movea.l cursoraddr(a5),a1
  386    000003A0 303C 008A               move.w  #138,d0
  387    000003A4 72F8                    moveq   #-8,d1
  388    000003A6 6100 FD5E               bsr     savecrtstate
  389    000003AA 12BC 0000               move.b  #0,(a1)
  390    000003AE D3FC 0000               adda.l  #width,a1
                0400
  391    000003B4 082B 0007 curchcheck    btst    #7,status(a3)   a3 setup by savecrtstate
                4001
  392    000003BA 67F8                    beq     curchcheck
  393    000003BC 12BC 0000               move.b  #0,(a1)
  394    000003C0 6100 FD6C               bsr     restcrtstate
  395    000003C4 4E75                    rts
  396
  397
  398    000003C6 7001          cshiftleft moveq   #1,d0         get pointer to last line of screen
  399    000003C8 D06D FFF4               add     maxy(a5),d0
  400    000003CC C0FC 4000               mulu    #16384,d0     16*1024*screenheight in d0
  401    000003D0 206D FFA6               movea.l screen(a5),a0
  402    000003D4 D1C0                    adda.l  d0,a0         pointer to last char line now in a0
  403    000003D6 2248                    movea.l a0,a1
  404    000003D8 5049                    addq    #8,a1         a1 will be source
  405    000003DA 70F8          cshift1   moveq   #-8,d0        get # pixels to move
  406    000003DC D06D FFF6               add     maxx(a5),d0
  407    000003E0 E748                    lsl     #3,d0         d0 has # pixels in keybuffer
  408    000003E2 4440                    neg     d0
  409    000003E4 780F                    moveq   #15,d4        counter for row move
  410    000003E6 3200                    move    d0,d1   set up width register
  411    000003E8 303C 0083               move    #131,d0 and replacement rule
  412    000003EC 6100 FD18               bsr     savecrtstate
  413    000003F0 1091          cshift3   move.b  (a1),(a0)     and go for it
  414    000003F2 D0FC 0400               adda    #width,a0     bump addresses
  415    000003F6 D2FC 0400               adda    #width,a1     to get next pixel row
  416    000003FA 082B 0007 cshift4       btst    #7,status(a3)    wait for move done
                4001
  417    00000400 67F8                    beq     cshift4
  418    00000402 51CC FFEC               dbra    d4,cshift3    count till 16 rows done
  419    00000406 6100 FD26               bsr   restcrtstate    fix replacement rule reg and return
  420    0000040A 4E75                    rts
  421
  422    0000040C 7001          cshiftright moveq  #1,d0         get pointer to last row
  423    0000040E D06D FFF4               add     maxy(a5),d0
```

```
00000412 C0FC 4000              mulu    #16384,d0
00000416 206D FFA6              movea.l screen(a5),a0
0000041A D1C0                   adda.l  d0,a0           a0 points to last char row
0000041C 2248                   movea.l a0,a1           make a copy
0000041E 5048                   addq    #8,a0           dest in a0 -- 1 char to right
00000420 60B8                   bra     cshift1         now do same stuff as shift left


                        *  procedure cexchange(savearea: windowp; ymin, ymax, xmin, width: shortint);

00000422 285F       cexchange   movea.l (sp)+,a4    a4 = return addr
00000424 301F                   move    (sp)+,d0    width of window in pixels in d0
00000426 E448                   lsr     #2,d0       d0=window width in long integers
00000428 5340                   subq    #1,d0       setup for later looping
0000042A 361F                   move    (sp)+,d3    d3 = x offset in chars
0000042C E74B                   lsl     #3,d3       d3 = x offset in pixels
0000042E 3A1F                   move    (sp)+,d5    d5 = ymax
00000430 321F                   move    (sp)+,d1    d1 = ymin
00000432 225F                   movea.l (sp)+,a1    a1 = ptr to save area
00000434 9941                   sub     d1,d5
00000436 5245                   addq    #1,d5       d5 has # of char rows to move
00000438 E94D                   lsl     #4,d5       now has # of pixel rows to move
0000043A 5345                   subq    #1,d5      setup for outer loop
0000043C C2FC 4000              mulu    #16384,d1   d1 = y offset into frame buffer
00000440 3800                   move    d0,d4       save d0 temporarily
00000442 7003                   moveq   #3,d0       setup replacement rule
00000444 6100 FCC0              bsr     savecrtstate
00000448 3004                   move    d4,d0      restore d0
0000044A 206D FFA6              movea.l screen(a5),a0 a0 points to frame buffer start
0000044E D1C1                   adda.l  d1,a0        now points to correct row
00000450 D0C3                   adda    d3,a0        do x offset into row
00000452 2448       cexchg2     movea.l a0,a2        make a working copy
00000454 3E00                   move    d0,d7        initialize inner loop
00000456 2C12       cexchg3     move.l  (a2),d6      screen to temp
00000458 24D1                   move.l  (a1),(a2)+   save area to screen
0000045A 22C6                   move.l  d6,(a1)+     temp to save area
0000045C 51CF FFF8              dbra    d7,cexchg3   inner loop (pixel row move)
00000460 D1FC 0000              adda.l  #width,a0    bump row pointer
                0400
00000466 51CD FFEA              dbra    d5,cexchg2   outer loop (row count)
0000046A 6100 FCC2              bsr     restcrtstate restore control regs
0000046E 4ED4                   jmp     (a4)         done


                        * procedure cscrollwindow( ymin, ymax, xmin, width: shortint);

00000470 6100 FF2A cscrollwindow bsr    changecursor
00000474 7400                   moveq   #0,d2           set upscroll flag in d2
00000476 285F       cscrollwindc movea.l (sp)+,a4       a4 = return addr
00000478 301F                   move    (sp)+,d0        d0 = width in chars
0000047A E748                   lsl     #3,d0           d0 = width in pixels
0000047C 4440                   neg     d0              setup for repl rule reg
0000047E 321F                   move    (sp)+,d1        d1 = x offset of window in chars
00000480 E749                   lsl     #3,d1           d1 = x offset in pixels (bytes)
00000482 3A1F                   move    (sp)+,d5        d5 = ymax
00000484 361F                   move    (sp)+,d3        d3 = ymin
00000486 9A43                   sub     d3,d5           d5 has # of rows to move
```



```
00000488 CAFC 0010              mulu    #16,d5          now d5 has # of pixel rows to move
0000048C 5345                   subq    #1,d5           setup for loop
0000048E 206D FFA6              movea.l screen(a5),a0   frame buffer addr in a0
00000492 C6FC 4000              mulu    #16384,d3       get y offset in bytes
00000496 D1C3                   adda.l  d3,a0           a0 points to first row of window
00000498 D0C1                   adda    d1,a0           now add in x offset
0000049A 4A42                   tst     d2              check up/down flag
0000049C 6650                   bne.s   cscrollwindb     and branch if dn
0000049E 2248                   movea.l a0,a1           make a copy for source pointer
000004A0 D3FC 0000              adda.l  #16384,a1       which starts 1 char row down
                4000
000004A6 3200                   move    d0,d1
000004A8 303C 0083              move    #131,d0   set up control regs
000004AC 6100 FC58              bsr     savecrtstate

000004B0 1091       cscrollwin1 move.b  (a1),(a0)       move a row
000004B2 D0FC 0400              adda    #width,a0
000004B6 D2FC 0400              adda    #width,a1
000004BA 082B 0007 cscrollwin2  btst    #7,status(a3)    a3 setup by savecrtstate
                4001
000004C0 67F8                   beq     cscrollwin2
000004C2 51CD FFEC              dbra    d5,cscrollwin1  loop till all rows moved
000004C6 7A0F       cscrollw2b  moveq   #15,d5          clear first or last line of window
000004C8 227C 000C              movea.l #clearl,a1
                C000
000004CE D3ED FFA6              adda.l  screen(a5),a1

000004D2 1091       cscrollwin3 move.b  (a1),(a0)       clear a pixel row
000004D4 D0FC 0400              adda    #width,a0
000004D8 082B 0007 cscrollwin4  btst    #7,status(a3)       wait for bitmover
                4001
000004DE 67F8                   beq     cscrollwin4
000004E0 51CD FFF0              dbra    d5,cscrollwin3  do 16 rows
000004E4 6100 FC48              bsr     restcrtstate
000004E8 2F0C                   move.l  a4,-(sp)        restack return addr
000004EA 6000 FEB0              bra     changecursor    and fixup cursor

000004EE 7800       cscrollwindb moveq  #0,d4           calculate first source row loc.
000004F0 3805                   move    d5,d4
000004F2 5244                   addq    #1,d4           d4 = #pixel rows to move
000004F4 740A                   moveq   #10,d2
000004F6 E5AC                   lsl.l   d2,d4           mpy by 1024 to get offset in FB
000004F8 D1C4                   adda.l  d4,a0           add to prev. calculated pointer
000004FA 90FC 0400              suba    #width,a0       point to bottom row to move
000004FE 2248                   movea.l a0,a1           a1 is source pointer
00000500 D0FC 4000              adda    #16384,a0       a0 points to destination
00000504 3200                   move    d0,d1           d1 has width for window reg
00000506 303C 0083              move    #131,d0         setup repl rule value
0000050A 6100 FBFA              bsr     savecrtstate
0000050E 1091       cscrollwin5 move.b  (a1),(a0)       move a pixel row
00000510 90FC 0400              suba    #width,a0       point to next src and dst
00000514 92FC 0400              suba    #width,a1
00000518 082B 0007 cscrollwin6  btst    #7,status(a3)   wait till bit mover done
                4001
0000051E 67F8                   beq     cscrollwin6
00000520 51CD FFEC              dbra    d5,cscrollwin5  go till all rows moved
00000524 2049                   movea.l a1,a0
```

```
  532     00000526 D0FC 0400               adda    #width,a0       a0 points to char row to clear
  533     0000052A 609A                    bra     cscroll1w2b
  534
  535     0000052C 6100 FE6E cscrollwinddn bsr     changecursor
  536     00000530 7401                    moveq   #1,d2           set down scroll flag
  537     00000532 6000 FF42               bra     cscrollwindc    go to common code
  538
  539
  540     00000536 6100 FE64 cdbscroll1    bsr     changecursor
  541     0000053A 7400                    moveq   #0,d2           set left scroll flag
  542     0000053C 285F      cdbscroll1b   movea.l (sp)+,a4        pickup return addr
  543     0000053E 321F                    move    (sp)+,d1        width in chars
  544     00000540 5341                    subq    #1,d1           actual width to move is 1 less
  545     00000542 E749                    lsl     #3,d1           width in pixels in d1
  546     00000544 4441                    neg     d1              setup for window width reg
  547     00000546 301F                    move    (sp)+,d0        x offset in chars
  548     00000548 E748                    lsl     #3,d0           d0 = x offset in pixels
  549     0000054A 3A1F                    move    (sp)+,d5        d5 = ymax
  550     0000054C 7600                    moveq   #0,d3
  551     0000054E 361F                    move    (sp)+,d3        d3 = ymin
  552     00000550 9A43                    sub     d3,d5
  553     00000552 5245                    addq    #1,d5           d5 = # char rows to move
  554     00000554 E94D                    lsl     #4,d5           d5 = # pixel rows to move
  555     00000556 5345                    subq    #1,d5           setup d5 for loop
  556     00000558 206D FFA6               movea.l screen(a5),a0
  557     0000055C 780E                    moveq   #14,d4
  558     0000055E E9AB                    lsl.l   d4,d3           d3 = d3*16384 ( y window start offset)
  559     00000560 D1C3                    adda.l  d3,a0
  560     00000562 D0C0                    adda    d0,a0           add in x offset
  561     00000564 2248                    movea.l a0,a1           copy to a1
  562     00000566 4A42                    tst     d2              check left/right flag
  563     00000568 6606                    bne.s   cdbscroll2      if right, skip
  564     0000056A D0FC 0008               adda    #8,a0           else src is to right of dest
  565     0000056E 6004                    bra.s   cdbscroll3
  566     00000570 D2FC 0008 cdbscroll2    adda    #8,a1           if right then src is left of dest
  567     00000574 303C 0083 cdbscroll3    move    #131,d0         setup replacement rule
  568     00000578 6100 FB8C               bsr     savecrtstate
  569     0000057C 1290      cdbscroll4    move.b  (a0),(a1)       move a pixel row
  570     0000057E D0FC 0400               adda    #width,a0        point to next row
  571     00000582 D2FC 0400               adda    #width,a1
  572     00000586 082B 0007 cdbscroll5    btst    #7,status(a3)   check status
          4001
  573     0000058C 67F8                    beq     cdbscroll5
  574     0000058E 51CD FFEC               dbra    d5,cdbscroll4   loop till all rows moved
  575     00000592 6100 FB9A               bsr     restcrtstate
  576     00000596 2F0C                    move.l  a4,-(sp)
  577     00000598 6000 FE02               bra     changecursor    finished!
  578
  579     0000059C 6100 FDFE cdbscroll1r   bsr     changecursor
  580     000005A0 7401                    moveq   #1,d2           set right shift flag
  581     000005A2 6098                    bra     cdbscroll1b     go to common code
  582
                           *   procedure cdbhighl(ord(char),x,y:shortint);
                           *
                           *   Assumes the character is in the highlight range
                           *   Does not know about current highlight state of character
                           *
```



```
  588     000005A4 285F      cdbhighl movea.l (sp)+,a4
  589     000005A6 301F               move    (sp)+,d0  d0 = y
  590     000005A8 C0FC 4000          mulu    #16384,d0
  591     000005AC 2040               movea.l d0,a0
  592     000005AE D1ED FFA6          adda.l  screen(a5),a0
  593     000005B2 3A1F               move    (sp)+,d5  d5 = x (this will be used later also)
  594     000005B4 E74D               lsl     #3,d5
  595     000005B6 D0C5               adda    d5,a0     a0 = address of byte to begin at
  596     000005B8 341F               move    (sp)+,d2    d2 = highlight char
  597     000005BA 6100 FDE0          bsr     changecursor    take off the cursor
  598     000005BE 303C 008A          move    #138,d0         repl rule = negate
  599     000005C2 72F8               moveq   #-8,d1          we will work with 8 byte wide chars
  600     000005C4 6100 FB40          bsr     savecrtstate
  601     000005C8 0802 0000          btst    #0,d2           invert?
  602     000005CC 6718               beq.s   cdbhigh3        no, try for underline
  603     000005CE 760F               moveq   #15,d3          setup loop for invert char
  604     000005D0 2248               movea.l a0,a1           copy pointer to the char
  605     000005D2 12BC 0000 cdbhigh1 move.b  #0,(a1)      do a row  RQ
  606     000005D6 D2FC 0400          adda    #width,a1       point to next row
  607     000005DA 082B 0007 cdbhigh2 btst    #7,status(a3)   is move done?
          4001
  608     000005E0 67F8               beq     cdbhigh2        wait here till done
  609     000005E2 51CB FFEE          dbra    d3,cdbhigh1     loop till 16 rows done
  610     000005E6 0802 0002 cdbhigh3 btst    #2,d2           underline?
  611     000005EA 6710               beq.s   cdbhigh5        no -- drop out
  612     000005EC D0FC 3C00          adda    #15360,a0       point a0 to last row of char
  613     000005F0 10BC 0000          move.b  #0,(a0)         and negate it
  614     000005F4 082B 0007 cdbhigh4 btst    #7,status(a3)   wait for done
          4001
  615     000005FA 67F8               beq     cdbhigh4
  616     000005FC 6100 FB30 cdbhigh5 bsr     restcrtstate
  617     00000600 2F0C               move.l  a4,-(sp)
  618     00000602 6000 FD98          bra     changecursor    put the cursor back
  619
  620                                 end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

Description

GLE_AUTL provides assembly language utility routines for the Graphics Low End (GLE).

```
PAGE   1 [3.0] 12/26/84 22:28:31  ASSEMBLY OF GLE_AUTL.TEXT                                  *** File name: GLE_AUTL ***

PASS 1 COMPLETE. ERRORS: 0
    1                       *
    2                       * Graphics Low End
    3                       *
    4                       * Module       = GLE_AUTL
    5                       * Programer    = BJS
    6                       * Date         = 9/30/82
    7                       *
    8                       * Purpose : To provide low level bit operations for GLE
    9                       *
   10                       *  Rev history
   11                       *
   12                       *    Created  - 9/30/82
   13                       *    Modified -
   14                       *
   15                       *
   16                       *     (c) Copyright Hewlett-Packard Company, 1983.
   17                       *     All rights are reserved.  Copying or other
   18                       *     reproduction of this program except for archival
   19                       *     purposes is prohibited without the prior
   20                       *     written consent of Hewlett-Packard Company.
   21                       *
   22                       *
   23                       *                RESTRICTED RIGHTS LEGEND
   24                       *
   25                       *     Use, duplication, or disclosure by the Government
   26                       *     is subject to restrictions as set forth in
   27                       *     paragraph (b) (3) (B) of the Rights in Technical
   28                       *     Data and Computer Software clause in
   29                       *     DAR 7-104.9(a).
   30                       *
   31                       *     HEWLETT-PACKARD COMPANY
   32                       *     Fort Collins, Colorado
   33                       *
   34                       *
   35                              MNAME GLE_AUTL
   36                              src MODULE GLE_AUTL;
   37                              src EXPORT
   38                              src FUNCTION GLE_IAND (VALUE1, VALUE2 : INTEGER) : INTEGER;
   39                              src FUNCTION GLE_IOR  (VALUE1, VALUE2 : INTEGER) : INTEGER;
   40                              src FUNCTION GLE_ISHIFT (VALUE, SHIFT : INTEGER) : INTEGER;
   41                              src END;
   42
   43    00000000                  rorg 0
   44                              def  GLE_AUTL_GLE_ISHIFT
   45                              def  GLE_AUTL_GLE_IAND
   46                              def  GLE_AUTL_GLE_IOR
   47                              def  GLE_AUTL_GLE_AUTL
   48
   49                              nosyms
   50
   51                      ********************************************************************************
   52                      *
   53                      *  T := GLE_ISHIFT ( VALUE, SHIFT )
   54                      *
   55                      *  This function shifts VALUE right (if SHIFT is negative) or left ( if
   56                      *  SHIFT is positive) by ABS(SHIFT).
   57                      *
   58             0000 0000 GLE_AUTL_GLE_ISHIFT equ *


PAGE   2 [3.0] 12/26/84 22:28:31  ASSEMBLY OF GLE_AUTL.TEXT                                  *** File name: GLE_AUTL ***

   59
   60    00000000 4E56 0000        link     a6,#0
   61
   62    00000004 202E 000C        move.l   12(a6),d0     value to shift
   63    00000008 222E 0008        move.l   8(a6),d1      shift value
   64
   65    0000000C 4A81             tst.l    d1
   66    0000000E 6D04             blt.s    shift_right
   67
   68    00000010 E3A0             asl.l    d1,d0
   69    00000012 6004             bra.s    shift_done
   70
   71             0000 0014 shift_right equ *
   72
   73    00000014 4481             neg.l    d1
   74    00000016 E2A0             asr.l    d1,d0
   75
   76             0000 0018 shift_done equ *
   77
   78    00000018 2D40 0010        move.l   d0,16(a6)
   79
   80    0000001C 4E5E             unlk     a6             return to pascal
   81    0000001E 205F             movea.l  (sp)+,a0
   82    00000020 504F             addq.w   #8,sp
   83    00000022 4ED0             jmp      (a0)
   84
   85                      ********************************************************************************
   86                      *
   87                      *  T := GLE_IAND ( VALUE1, VALUE2 )
   88                      *
   89                      *  This function and's VALUE1 with VALUE2.
   90                      *
   91             0000 0024 GLE_AUTL_GLE_IAND equ *
   92
   93    00000024 4E56 0000        link     a6,#0
   94
   95    00000028 202E 000C        move.l   12(a6),d0     value2
   96    0000002C 222E 0008        move.l   8(a6),d1      value1
   97
   98    00000030 C081             and.l    d1,d0         perform operation
   99
  100    00000032 2D40 0010        move.l   d0,16(a6)     put result on stack
  101
  102    00000036 4E5E             unlk     a6             return to pascal
  103    00000038 205F             movea.l  (sp)+,a0
  104    0000003A 504F             addq.w   #8,sp
  105    0000003C 4ED0             jmp      (a0)
  106
  107                      ********************************************************************************
  108                      *
  109                      *  T := GLE_IOR ( VALUE1, VALUE2 )
  110                      *
  111                      *  This function OR's VALUE1 with VALUE2.
  112                      *
  113             0000 003E GLE_AUTL_GLE_IOR equ *
  114
  115    0000003E 4E56 0000        link     a6,#0
```

```
  116
  117    00000042 202E 000C        move.l   12(a6),d0      value2
  118    00000046 222E 0008        move.l   8(a6),d1       value1
  119
  120    0000004A 8081             or.l     d1,d0          perform operation
  121
  122    0000004C 2D40 0010        move.l   d0,16(a6)      put result on stack
  123
  124    00000050 4E5E             unlk     a6             return to pascal
  125    00000052 205F             movea.l  (sp)+,a0
  126    00000054 504F             addq.w   #8,sp
  127    00000056 4ED0             jmp      (a0)
  128
  129    00000058 4E75      GLE_AUTL_GLE_AUTL RTS          module init procedure
  130                         END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

Description

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
    2                       ********************************************************************************
    3                       *
    4                       *       COPYRIGHT (C) 1984 BY HEWLETT-PACKARD COMPANY
    5                       *
    6                       ********************************************************************************
    7                       *
    8                       *
    9                       *       IOLIB     EXTG
   10                       *
   11                       *
   12                       ********************************************************************************
   13                       *
   14                       *
   15                       *
   16                       *       Library - IOLIB
   17                       *       Author  -
   18                       *       Phone   -
   19                       *
   20                       *       Purpose - This set of assembly language code is intended to be used as
   21                       *                 a PASCAL module for I/O drivers for use by the external I/O
   22                       *                 procedures library.
   23                       *
   24                       *                 Much of this code was taken from HPL GPIO
   25                       *                 drivers.
   26                       *
   27                       *       Date    - 09/20/81
   28                       *       Update  - 04/26/84
   29                       *       Release - 05/15/84
   30                       *
   31                       *
   32                       *       Source  - IOLIB:GPIO.TEXT
   33                       *       Object  - IOLIB:GPIO.CODE
   34                       *
   35                       *
   36                       ********************************************************************************
   37                       *
   38                       *
   39                       *       RELEASED
   40                       *       VERSION         3.0
   41                       *
   42                       *
   43                       ********************************************************************************
```



```
   45                       ********************************************************************************
   46                       *                                                                              *
   47                       *                                                                              *
   48                       *       BUG FIX HISTORY         - after release 1.0                            *
   49                       *                                                                              *
   50                       *                                                                              *
   51                       *       BUG #   BY  / ON        LOC             DESCRIPTION                    *
   52                       *       -----   -----------     --------------  ----------------------         *
   53                       *                                                                              *
   54                       *       SPR836  -----------     G_SET_PCT       a IOCONTROL(x,1,1); does       *
   55                       *               08/09/1982                      not set the PCTL line, it      *
   56                       *                                               resets the interface.          *
   57                       *                                                                              *
   58                       *       475     -----------     all over        Change BSRs into JSRs to       *
   59                       *               09/17/1982                      allow re-placement of the      *
   60                       *                                               modules.  Also in HPIB and     *
   61                       *                                               Data comm modules.             *
   62                       *                                                                              *
   63                       *       507     -----------     G_TFI           Did not do a trigger to the    *
   64                       *               09/17/1982                      gpio card before a FHS tfr.    *
   65                       *                                                                              *
   66                       *       508     -----------     G_WTC           Typographical error in         *
   67                       *               09/17/1982                      write control to CTL0/1.       *
   68                       *                                                                              *
   69                       *       xxx     -----------     G_TFR           Transfer should wait for       *
   70                       *               10/06/1982                      card to be ready ( ala the     *
   71                       *                                               HPL system ).                  *
   72                       *                                                                              *
   73                       *       yyy     ---------       G_INIT          Changes for timing on          *
   74                       *               8/1/83          G_WAIT          680xx UMM CPU boards           *
   75                       *               5/2/84          G_STSCHK                                       *
   76                       *                                                                              *
   77                       *        69     ---------       G_WTC           Missing inst. in set CTL0/1    *
   78                       *     (3.0 QA)  4/26/84                                                        *
   79                       *                                                                              *
   80                       *     SPR12724  ---------       G_TFR           Clear upper byte on byte       *
   81                       *               5/3/84                          transfers                      *
   82                       ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *       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
                                REFA CHECK_TIMER                                        yyyy JS 8/ 1/83
                                REFA DELAY_TIMER                                        yyyy JS 8/ 1/83
                        *
                        *       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
                                LMODE CHECK_TIMER                                       yyyy JS 8/ 1/83
                                LMODE DELAY_TIMER                                       yyyy JS 8/ 1/83
```

```
PAGE   5 [3.0] 12/26/84 21:31:49 IOLIB EXTG - COMMON DECLARATIONS AND EQUATES              *** File name: GPIO ***
  177                                 INCLUDE COMDCL
```

```
PAGE   6 [3.0] 12/26/84 21:31:49 IOLIB COMDCL - COMMON EQUATES AND DEFINITIONS            *** File name: GPIO ***
  180                         ********************************************************************************
  181                         *
  182                         *       modified:  02/22/82  JPC    added parm to user EOT & ISR proc's
  183                         *                  08/01/83  JS     added timer_present and sysflag2 equ's
  184                         *
  185                         ********************************************************************************
  186                         *
  187                         *       HPL CONVENTIONS
  188                         *
  189                         *
  190                         *         Much of this code is taken intact from the 9826 HPL
  191                         *         language system EIO ROM ( extended I/O ROM ).
  192                         *
  193                         *         The Pascal that will be calling this code uses
  194                         *         the stack for parameter passage.  The HPL code
  195                         *         uses the Ax and Dx registers for all parameters.
  196                         *         The Pascal driver entry points on the previous pages
  197                         *         take care of getting the parameters into the correct
  198                         *         registers.
  199                         *
  200                         *
  201                         *       GENERAL HPL ENTRY/EXIT CONDITIONS:
  202                         *
  203                         *         A1.L = CARD ADDRESS
  204                         *         A2.L = DRIVER TEMP ADDRESS
  205                         *         UNLESS OTHERWISE INDICATED, THESE REGISTERS ARE UNALTERED.
  206                         *
  207                         *
  208                         *       NEW ENTRY/EXIT CONDITIONS FOR PASCAL USE :
  209                         *
  210                         *         A3.L = BUFFER CONTROL BLOCK ADDRESS
  211                         *                In addition to the A1/A2 convention, Pascal will use
  212                         *                A3 for a pointer to the buffer control block.
  213                         *                The HPL system kept much of the transfer
  214                         *                information in the s.c. temps.
  215                         *
  216                         *         TIMEOUT(A2) = contains timeout information
  217                         *                Timeout was a global temp in HPL and a timeout
  218                         *                generated an error.
  219                         *                In PASCAL each card has a timeout value stored in
  220                         *                its temporary area.  A timeout error
  221                         *                generates an ESCAPE ( which can be trapped ).
  222                         *
  223                         *
  224                         ********************************************************************************
```

```
                        ********************************************************************************
                        *
                        *
                        *         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

              0000 0001 TIMER_PRESENT  EQU   1        JS  8/1/83  SYSFLAG2 BIT -- 0=>TIMER PRESENT
              FFFF FEDA SYSFLAG2       EQU   $FFFFFEDA           JS  8/1/83

                        ********************************************************************************
                        *
```



```
                        *         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 0576          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 04FE          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 00EE          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 00CE             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 00CA             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 00AA             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 0204             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 024A             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 02A4             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 0158             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 016C             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 0178             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 0000 G_RESET  JSR     ABORT_IO                                    475 TM 9/17/82
000000E8 357C FFFF 0002          MOVE.W  #-1,MA_W(A2)   the card is not addressable - so my addr =-1
000000EE 1340 0001               MOVE.B  D0,1(A1)       Reset the GPIO card
000000F2 2F3C 0000 0028          MOVE.L  #40,-(SP)         Wait AT LEAST 15 microseconds
000000F8 4EB9 0000 0000          JSR     DELAY_TIMER    USE DELAY ROUTINE 40 uS       yyyy JS 8/1/83
000000FE 137C 0003 0002          MOVE.B  #3,2(A1)
00000104 4E75                    RTS
                        ****************************************
                        *
                        *  IOFS
                        *
                        ****************************************
00000106 7000           G_IOFS   MOVEQ   #0,D0
00000108 0829 0003 0007          BTST    #3,7(A1)     PSTS -> D0<0>
0000010E 6702                    BEQ.S   G_IOFS1
00000110 7001                    MOVEQ   #1,D0
00000112 0811 0000 G_IOFS1  BTST    #0,(A1)      PFLG -> D0<1>
00000116 6702                    BEQ.S   G_IOFS2
00000118 5440                    ADDQ    #2,D0
0000011A 4E75           G_IOFS2  RTS
```



```
                        ****************************************
                        *
                        *  wtb
                        *
                        ****************************************
0000011C 6110           G_WTB    BSR.S   G_STSCHK     Check for PSTS
0000011E 616E                    BSR.S   G_WAIT       Wait for FLG
00000120 3340 0004               MOVE.W  D0,4(A1)     Write the word
00000124 1280                    MOVE.B  D0,(A1)      wti 7,0  trigger output
00000126 4E75                    RTS
                        ****************************************
                        *
                        *  rdb
                        *
                        ****************************************
00000128 6110           G_RDB    BSR.S   G_STSCHK     Check for PSTS
0000012A 6162                    BSR.S   G_WAIT
0000012C 3029 0004               MOVE.W  4(A1),D0     Set I/O line for input
00000130 1280                    MOVE.B  D0,(A1)      wti 7,0  trigger input
00000132 615A                    BSR.S   G_WAIT       Wait for data to come in
00000134 3029 0004               MOVE.W  4(A1),D0
00000138 4E75                    RTS
0000013A 0829 0003 0007 G_STSCHK BTST    #3,7(A1)     CHECK FOR PSTS OK     yyyy JS 8/1/83
00000140 6618                    BNE.S   G_STSRTS     IF OK THEN RETURN     yyyy JS 8/1/83
00000142 0838 0001 FEDA          BTST    #TIMER_PRESENT,SYSFLAG2   TIMER EXISTS?  yyyy JS 8/1/83
00000148 671A                    BEQ.S   G_STS0B      YES, USE IT           yyyy JS 8/1/83
0000014A 3F00                    MOVE    D0,-(SP)     SAVE D0               yyyy JS 8/1/83
0000014C 303C 55F0               MOVE    #22000,D0    SETUP 100MS LOOP      yyyy JS 8/1/83
00000150 0829 0003 0007 G_STS0   BTST    #3,7(A1)     Is PSTS O.K. ?
00000156 6704                    BEQ.S   G_STS1       No:  Give him 100 ms
00000158 301F                    MOVE    (SP)+,D0     Yes: Restore D0
0000015A 4E75           G_STSRTS RTS
0000015C 51C8 FFF2 G_STS1   DBRA    D0,G_STS0    Decrement loop counter
00000160 6000 008C               BRA     CRD_DOWN     If counter expires, card is down
00000164 1F3C 0001 G_STS0B  MOVE.B  #1,-(SP)     SETUP TIMER RECORD    yyyy JS 8/1/83
00000168 2F3C 0000 0064          MOVE.L  #100,-(SP)   LOOP FOR 100 MS       yyyy JS 8/1/83
0000016E 0829 0003 0007 G_STS0C  BTST    #3,7(A1)     PSTS SET ?            yyyy JS 8/1/83
00000174 6614                    BNE.S   G_STS1B      YES, RETURN           yyyy JS 8/1/83
00000176 4857                    PEA     (SP)         CHECK TIMER           yyyy JS 8/1/83
00000178 4EB9 0000 0000          JSR     CHECK_TIMER  SEE IF 100MS GONE     yyyy JS 8/1/83
0000017E 6AEE                    BPL     G_STS0C      NO, CHECK PSTS        yyyy JS 8/1/83
```

```
  578    00000180 0829 0003          BTST    #3,7(A1)      LAST CHANCE TEST     yyyy JS 5/2/84
                  0007
  579    00000186 6700 0066          BEQ     CRD_DOWN      IF NOT PSIS, ERROR   yyyy JS 5/2/84
  580    0000018A 5C4F       G_STS1B ADDQ    #6,SP         STRIP TIMER REC      yyyy JS 8/1/83
  581    0000018C 4E75               RTS                   AND RETURN           yyyy JS 8/1/83
  582
```



```
  584                         ******************************************
  585                         *
  586                         *  GPIO WAIT ROUTINE
  587                         *
  588                         ******************************************
  589    0000018E 0811 0000 G_WAIT   BTST    #0,(A1)       IF CARD_READY THEN RETURN
  590    00000192 662A               BNE.S   G_WRTS
  591    00000194 222A 002E          MOVE.L  TIMEOUT(A2),D1 D1 = (TIMEOUT)
  592    00000198 671E               BEQ.S   G_WAIT4
  593    0000019A 0838 0001          BTST    #TIMER_PRESENT,SYSFLAG2  GOT A TIMER?    yyyy JS 8/1/83
                  FEDA
  594    000001A0 671E               BEQ.S   G_WAIT2B              YES, SO USE IT     yyyy JS 8/1/83
  595              0000 01A2 G_WAIT2  EQU     *-
  596                         * tm    MULU    #182,D1       CONVERT D1 TO MILLISECONDS
  597    000001A2 2A01               MOVE.L  D1,D5         \
  598    000001A4 EF89               LSL.L   #7,D1          \   * 196 IS CLOSE TO * 182
  599    000001A6 ED8D               LSL.L   #6,D5          /
  600    000001A8 D285               ADD.L   D5,D1         /
  601    000001AA 0811 0000 G_WAIT3  BTST    #0,(A1)       WHILE D1 MILLISECONDS HAS NOT EXPIRED DO
  602    000001AE 660E               BNE.S   G_WRTS        IF CARD_READY THEN RETURN
  603    000001B0 5381               SUBQ.L  #1,D1
  604    000001B2 6EF6               BGT.S   G_WAIT3
  605    000001B4 7011       G_WAITER MOVEQ  #TMO_ERR,D0   GENERATE ESCAPE
  606    000001B6 6044               BRA.S   ESC_ERR
  607
  607
  607
  607
  608    000001B8 0811 0000 G_WAIT4  BTST    #0,(A1)       IF CARD_READY THEN RETURN
  609    000001BC 67FA               BEQ.S   G_WAIT4
  610    000001BE 4E75       G_WRTS  RTS
  611
  612    000001C0 1F3C 0001 G_WAIT2B MOVE.B  #1,-(SP)       SETUP TIMER RECORD          yyyy JS 8/1/83
  613    000001C4 2F01               MOVE.L  D1,-(SP)      D1 HAS MS TO WAIT           yyyy JS 8/1/83
  614    000001C6 223C 0000 G_WAIT3B MOVE.L  #364,D1       INNER LOOP TO GET 1 MS AT   yyyy JS 8/1/83
                  016C
  615    000001CC 0811 0000 G_WAIT4B BTST    #0,(A1)       16 MHZ -- CHECK FOR READY   yyyy JS 8/1/83
  616    000001D0 6614               BNE.S   G_WAIT5B      READY -- EXIT THE LOOP      yyyy JS 8/1/83
  617    000001D2 5381               SUBQ.L  #1,D1         ELSE LOOP FOR 1-2 MS HERE   yyyy JS 8/1/83
  618    000001D4 6EF6               BGT     G_WAIT4B                                  yyyy JS 8/1/83
  619    000001D6 4857               PEA     (SP)          AFTER INNER LOOP CHECK TIMER yyyy JS 8/1/83
  620    000001D8 4EB9 0000          JSR     CHECK_TIMER                               yyyy JS 8/1/83
                  0000
  621    000001DE 6AE6               BPL     G_WAIT3B      BRANCH IF NO TIMEOUT        yyyy JS 8/1/83
  622    000001E0 0811 0000          BTST    #0,(A1)       TIMEOUT -- LAST CHANCE      yyyy JS 5/2/84
  623    000001E4 67CE               BEQ     G_WAITER      REAL TIMEOUT IF NOT READY   yyyy JS 5/2/84
  624    000001E6 5C4F       G_WAIT5B ADDQ   #6,SP          CLEAN UP STACK              yyyy JS 8/1/83
  625    000001E8 4E75               RTS                   DONE                        yyyy JS 8/1/83
  626
```

```
                        ********************************************************************************
                        *
                        *       Error escapes
                        *
                        ********************************************************************************
000001EA 7008   G_SCBSY   MOVEQ  #SC_BUSY,D0          sc is busy
000001EC 600E             BRA.S  ESC_ERR
000001EE 7015   CRD_DOWN  MOVEQ  #CRD_DWN,D0          CARD IS DOWN
000001F0 600A             BRA.S  ESC_ERR
000001F2 7014   G_SC_ERR  MOVEQ  #BAD_SCT,D0          bad set/clear/test
000001F4 6006             BRA.S  ESC_ERR
000001F6 7007   GTERR_B   MOVEQ  #TFR_ERR,D0          bad transfer specification
000001F8 6002             BRA.S  ESC_ERR
000001FA 700D   GTERR_D   MOVEQ  #NO_DMA,D0           DMA not installed
                *         BRA.S  ESC_ERR


000001FC 48C0   ESC_ERR   EXT.L  D0                   assume errors<128
000001FE 2B40 FFBE        MOVE.L D0,IOE_RSLT(A5)      save ioe_result
00000202 102A 002D        MOVE.B IO_SC(A2),D0
00000206 2B40 FFBA        MOVE.L D0,IOE_SC(A5)        save io s.c.
0000020A 3B7C FFE6        MOVE.W #IOE_ERROR,ESC_CODE(A5)
         FFFE
00000210 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
                        *
                        ********************************************
00000212 C23C 0007 G_SET    AND.B  #7,D1            MASK TO RIGHT SIZE
00000216 B23C 0005          CMP.B  #5,D1            \ HANDLE SET PCTL
0000021A 6714               BEQ.S  GSET_1           /
0000021C B23C 0006          CMP.B  #6,D1            \ CAN ONLY SET
00000220 6DD0               BLT.S  G_SC_ERR         /     CTL0/1
00000222 923C 0005          SUB.B  #5,D1            GET INTO 1/2 FORM
00000226 8329 0007          OR.B   D1,7(A1)         SET CTL0 or 1
0000022A 832A 002C          OR.B   D1,EIRB_OFF(A2)  SAVE IN EIRB
0000022E 4E75               RTS
00000230 1341 0000 GSET_1   MOVE.B D1,0(A1)         SET PCTL
00000234 4E75               RTS


                        ********************************************
                        *
                        *       CLEAR LINE
                        *
                        ********************************************
00000236 C23C 0007 G_CLR    AND.B  #7,D1            MASK TO RIGHT SIZE
0000023A B23C 0006          CMP.B  #6,D1            \ CAN ONLY CLEAR
0000023E 6DB2               BLT.S  G_SC_ERR         /     CTL0/1
00000240 102A 002C          MOVE.B EIRB_OFF(A2),D0  GET OLD CTL0/1
00000244 923C 0006          SUB.B  #6,D1            GET INTO 0/1 FORM
00000248 0380               BCLR   D1,D0            CLEAR COPY
0000024A 1540 002C          MOVE.B D0,EIRB_OFF(A2)  SAVE IN EIRB
0000024E 1340 0007          MOVE.B D0,7(A1)         WRITE TO CARD
00000252 4E75               RTS


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

```
                        ********************************************
00000254 4240       G_TEST   CLR.W   D0              set FALSE indication
00000256 C23C 0007           AND.B   #7,D1           mask to the right size
0000025A B23C 0007           CMP.B   #7,D1           \ CHECK FOR PCTL
0000025E 6792                BEQ     G_SC_ERR        /       AND GIVE ERROR
00000260 B23C 0003           CMP.B   #3,D1
00000264 6E0A                BGT.S   GTST2
00000266 0329 0007 GTST1     BTST    D1,7(A1)        TEST STO/1,EIR,or psts
0000026A 6702        GTST_CHK BEQ.S  GTST_RTS        if clear then RTS
0000026C 7001        GTST_SET MOVEQ  #1,D0              else return true indication
0000026E 4E75        GTST_RTS RTS

00000270 923C 0004 GTST2     SUB.B   #4,D1
00000274 B23C 0001           CMP.B   #1,D1
00000278 6E06                BGT.S   GTST4
0000027A 032A 002C GTST3     BTST    D1,EIRB_OFF(A2) TEST CTL0/1
0000027E 60EA                BRA.S   GTST_CHK        GO TO COMMON CHECK CODE

00000280 0829 0000 GTST4     BTST    #0,0(A1)        CHECK READY LINE
         0000
00000286 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
                        *
                        *
00000288 41FA 004E G_RDS     LEA     G_RDSTBL,A0            get pointer to lookup table
0000028C D241                ADD.W   D1,D1                  multiply the rds register by 2
0000028E B23C 000C           CMP.B   #G_RT_SIZ,D1           \ check for out of bounds
00000292 6C18                BGE.S   RDS_ERR                /
00000294 3030 1000           MOVE.W  0(A0,D1),D0            get the table entry
00000298 6B12                BMI.S   RDS_ERR                if the entry is 0 then error
0000029A B03C 0001           CMP.B   #G_TEMP,D0
                        * tm    BEQ.S   GR_TEMP
0000029E 6D12                BLT.S   GR_CARD
000002A0 E048                LSR     #8,D0                  get the routine offset
000002A2 671A                BEQ.S   G_RDS_DI               -  status rtn 3 - data in
000002A4 5340                SUBQ    #1,D0
000002A6 671C                BEQ.S   G_RDS_RDY              -  status 4 - ready
000002A8 5340                SUBQ    #1,D0
000002AA 6722                BEQ.S   G_RDS_PST              -  status  5 - peripheral status
                        *       BRA.S   RDS_ERR                there are no more status 'routines'


000002AC 7013        RDS_ERR MOVEQ   #BAD_RDS,D0            bad read status
000002AE 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
                        *
000002B2 E048        GR_CARD  LSR     #8,D0                  get the card offset
000002B4 1031 0000 GR_CARD1 MOVE.B 0(A1,D0),D0              get the value
000002B8 0240 00FF GREGEXIT ANDI.W #$00FF,D0                mask off garbage
000002BC 4E75               RTS


                        *
                        *       data in
```

```
PAGE  21 [3.0] 12/26/84 21:31:49 IOLIB EXTG - GPIO DRIVERS                    *** File name: GPIO ***

 776                        *
 777    000002BE 3029 0004 G_RDS_DI  MOVE.W 4(A1),D0              get data lines
 778    000002C2 4E75                RTS
 779
 779
 779
 780    000002C4 1029 0000 G_RDS_RDY MOVE.B 0(A1),D0              get ready line
 781    000002C8 C200 0001           ANDI.B #$01,D0               mask to 1 bit
 782    000002CC 60EA                BRA    GREGEXIT              and get out
 783
 783
 783
 784    000002CE 1029 0007 G_RDS_PST MOVE.B 7(A1),D0              get status
 785    000002D2 C200 000F           ANDI.B #$0F,D0               mask to 4 bits
 786    000002D6 60E0                BRA    GREGEXIT              and get out
 787
 787
 787
 788              0000 02D8 G_RDSTBL  EQU    *
 789    000002D8 01                  DC.B   1,G_CRDREG           status 0 - card reg  0   - card id
 789    000002D9 00
 790    000002DA 03                  DC.B   3,G_CRDREG           status 1 - card reg 3    - intr/dma status
 790    000002DB 00
 791    000002DC 09                  DC.B   9,G_ROUTINE          status 2 - not implemented
 791    000002DD 02
 792    000002DE 00                  DC.B   0,G_ROUTINE          status 3 - data in
 792    000002DF 02
 793    000002E0 01                  DC.B   1,G_ROUTINE          status 4 - ready line
 793    000002E1 02
 794    000002E2 02                  DC.B   2,G_ROUTINE          status 5 - peripneral status
 794    000002E3 02
 795              0000 02E4 G_RT_END  EQU    *
 796              0000 000C G_RT_SIZ  EQU    G_RT_END-G_RDSTBL   size of table
```

```
PAGE  22 [3.0] 12/26/84 21:31:49 IOLIB EXTG - GPIO DRIVERS                    *** File name: GPIO ***

 798                        ********************************************************************************
 799                        *
 800                        *       G_WTC
 801                        *
 802                        *         WRITE CONTROL
 803                        *
 804                        *         ENTRY:  D0.W = PARAMETER
 805                        *
 806                        ********************************************************************************
 807    000002E4 B27C 0006 G_WTC     CMP.W  #6,D1                 \
 808    000002E8 6CC2                BGE.S  RDS_ERR               / check wtc limits
 809    000002EA 48C1                EXT.L  D1
 810    000002EC D281                ADD.L  D1,D1
 811    000002EE 4EFB 1002           JMP    GWTCTBL(D1)
 812
 812
 813    000002F2 600A      GWTCTBL   BRA.S  G_WTC_RST             CONTROL 0 - DO A RESET
 814    000002F4 600C                BRA.S  G_SET_PCT             CONTROL 1 - set pctl
 815    000002F6 6010                BRA.S  G_SET_CTL             CONTROL 2 : set control lines
 816    000002F8 6022                BRA.S  G_DATA_O              CONTROL 3 - write data out
 817    000002FA 60B0                BRA.S  RDS_ERR               CONTROL 4 : not used
 818    000002FC 6024                BRA.S  G_EIR                 CONTROL 5 : enable intrpts
 819
 819
 819
 820    000002FE 6000 FDE2 G_WTC_RST BRA    G_INIT                reset card
 821
 821
 822    00000302 1340 0000 G_SET_PCT MOVE.B D0,0(A1)               set pctl line          SPR836 TM 8/9/82
 823    00000306 4E75                RTS
 824
 824
 825    00000308 1340 0007 G_SET_CTL MOVE.B D0,7(A1)               set ctl 0 and 1
 826    0000030C C03C 0003           AND.B  #3,D0                 \ save CTL0/1          508 TM 9/17/82
 827    00000310 022A 00FC           ANDI.B #$FC,EIRB_OFF(A2)     CLEAR CTL0/1     #69 (3.0) JWS 4/26/84
                 002C
 828    00000316 812A 002C           OR.B   D0,EIRB_OFF(A2)       / in EIR byte          508 TM 9/17/82
 829    0000031A 4E75                RTS
 830    0000031C 3340 0004 G_DATA_O  MOVE.W D0,4(A1)               write 16 bit data
 831    00000320 4E75                RTS
```

```
                        ************************************************
                        *
                        *  EIR
                        *
                        ************************************************
00000322 1540 002C G_EIR    MOVE.B  D0,EIRB_OFF(A2)
00000326 4EB9 0000          JSR     ITXFR                                   475 TM 9/17/82
         0000
0000032C 6616               BNE.S   G_RTS                   if tfr then don't
0000032E 1340 0007 G_WTI5   MOVE.B  D0,7(A1)                Update CTL1:CTL0
00000332 0800 0005          BTST    #5,D0                   Check RESET bit
00000336 6600 FDAA          BNE     G_RESET
0000033A 4A00               TST.B   D0
0000033C 6C08               BGE.S   G_RTS1
0000033E 137C 0080          MOVE.B  #$80,3(A1)
         0003
00000344 4E75       G_RTS   RTS
00000346 137C 0000 G_RTS1   MOVE.B  #0,3(A1)
         0003
0000034C 4E75               RTS
```



```
                        ************************************************
                        *
                        *  GPIO tfr
                        *
                        ************************************************
0000034E 4EB9 0000 G_TFR    JSR     CHECK_TFR               wait for tfr to finish  475 TM 9/17/82
         0000
00000354 6100 FE38          BSR     G_WAIT                  Wait for FLG            xxx TM 10/6/82
00000358 337C 0000          MOVE.W  #0,4(A1)                CLEAR DATA REG    SPR12724  JS 5/3/84
         0004
0000035E 4A2B 000B          TST.B   TEND_OFF(A3)            \ end NOT ALLOWED
00000362 6600 FE92          BNE     GTERR_B                 /
00000366 202B 0010          MOVE.L  TCNT_OFF(A3),D0         GET COUNT
0000036A 4241               CLR.W   D1                      \
0000036C 122B 0009          MOVE.B  TUSR_OFF(A3),D1         \   COMPUTE OFFSET INTO JUMP TABLE
00000370 D241               ADD.W   D1,D1                    \
00000372 4EB9 0000          JSR     TESTDMA                  /  BASED ON TFR       475 TM 9/17/82
         0000
00000378 6704               BEQ.S   G_NODMA                 /      TYPE AND DMA PRESENCE
0000037A 0641 0014          ADDI.W  #20,D1                  /
0000037E 41FA 0008 G_NODMA  LEA     G_TBL,A0                \
00000382 D0F0 1000          ADDA.W  0(A0,D1),A0              INDEXED JUMP THRU TABLE
00000386 4ED0               JMP     (A0)                    /
                        *
                        *       TRANSFER JUMP TABLE
                        *
                        *                                   --------------------  DMA is not installed or available
00000388 FE6E       G_TBL   DC.W    GTERR_B-G_TBL   serial  interrupt
0000038A FE72               DC.W    GTERR_D-G_TBL   serial  dma
0000038C 00F4               DC.W    G_T_FHS-G_TBL   serial  fhs
0000038E 00F4               DC.W    G_T_FHS-G_TBL   serial  fastest
00000390 FE6E               DC.W    GTERR_B-G_TBL   serial  overlap
                        *                               --------------------
00000392 00C0               DC.W    G_T_INT-G_TBL   overlap interrupt
00000394 FE72               DC.W    GTERR_D-G_TBL   overlap dma
00000396 00C8               DC.W    G_T_BST-G_TBL   overlap fhs
00000398 00C8               DC.W    G_T_BST-G_TBL   overlap fastest
0000039A 00C0               DC.W    G_T_INT-G_TBL   overlap overlap
                        *                               --------------------  DMA is installed
0000039C FE6E               DC.W    GTERR_B-G_TBL   serial  interrupt
0000039E 0028               DC.W    G_T_DMA-G_TBL   serial  dma
000003A0 00F4               DC.W    G_T_FHS-G_TBL   serial  fhs
000003A2 0028               DC.W    G_T_DMA-G_TBL   serial  fastest
000003A4 FE6E               DC.W    GTERR_B-G_TBL   serial  overlap
                        *                               --------------------
000003A6 00C0               DC.W    G_T_INT-G_TBL   overlap interrupt
000003A8 0028               DC.W    G_T_DMA-G_TBL   overlap dma
000003AA 00C8               DC.W    G_T_BST-G_TBL   overlap fhs
000003AC 0028               DC.W    G_T_DMA-G_TBL   overlap fastest
000003AE 0028               DC.W    G_T_DMA-G_TBL   overlap overlap
```

```
                        *
                        *       Transfer DMA
                        *
000003B0 B0BC 0000 G_T_DMA  CMP.L  #1,D0              \ USE INTR IF COUNT=1 ON DMA
                 0001
000003B6 6700 0090          BEQ    G_T_INT          /
000003BA 177C 0002          MOVE.B #TT_DMA,TACT_OFF(A3)  set tfr type to DMA
                 0007
000003C0 4A2B 000D          TST.B  TDIR_OFF(A3)     \ test for transfer direction
000003C4 6E36               BGT.S  G_TOD            /
                        *
                        *       Transfer Input Dma:
                        *
            0000 03C6 G_TID     EQU    *
000003C6 5380               SUBQ.L #1,D0            0 - Set up DMA for Input
000003C8 4EB9 0000          JSR    GETDMA           Tfr N-1 bytes via DMA   475 TM 9/17/82
                 0000
000003CE 0829 0003          BTST   #3,(A1)          Is BURST bit set on GPIO card ?
                 0003
000003D4 6704               BEQ.S  G_DMA1           No
000003D6 0042 0008          ORI.W  #8,D2            Yes: Set BURST bit in DMA arm byte
000003DA 3882       G_DMA1  MOVE   D2,(A4)          Arm DMA
000003DC 6100 005E          BSR    GD_STBSY         Set buffer busy (Card is still not triggered)
000003E0 4A69 0004          TST    4(A1)            rdi 4
000003E4 1280               MOVE.B D0,(A1)          wti 7,0  (trigger card)
000003E6 137C 0001          MOVE.B #1,2(A1)         RDYEN = 0 (so transfers won't interrupt)
                 0002
000003EC 4A2B 000A          TST.B  T_BW_OFF(A3)     Byte (0) or Word (1) transfers ?
000003F0 6704               BEQ.S  G_DMA3
000003F2 D67C 0004          ADD    #4,D3
000003F6 1343 0003 G_DMA3   MOVE.B D3,3(A1)         Tell the GPIO what channel he's got and
                        *                             watch the shit hit the fan
000003FA 602C               BRA.S  G_DMA_W          DONE
                        *
                        *       Transfer Output Dma:
                        *
            0000 03FC G_TOD     EQU    *
000003FC 4EB9 0000 G_DMAOUT JSR    GETDMA           Get a DMA channel       475 TM 9/17/82
                 0000
00000402 0829 0003          BTST   #3,(A1)          Is Burst mode enabled (1)
                 0003
00000408 6704               BEQ.S  G_DMA4
0000040A 0042 0008          ORI.W  #8,D2            Yes: Set the Burst bit in DMA arm byte
0000040E 3882       G_DMA4  MOVE   D2,(A4)          Arm the DMA channel
00000410 6100 002A          BSR    GD_STBSY         Set the buffer busy
00000414 137C 0001          MOVE.B #1,2(A1)         Disable the transfer interrupt mechanism
                 0002
0000041A 4A2B 000A          TST.B  T_BW_OFF(A3)     Byte (1) or Word (0) transfers ?
0000041E 6704               BEQ.S  G_DMA6           Byte: Don't set the WORD bit on GPIO card
00000420 0043 0004          ORI.W  #4,D3            WORD: Set the WORD bit
00000424 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
                        *
00000428 182B 0009 G_DMA_W  MOVE.B TUSR_OFF(A3),D4  \
0000042C B83C 0005          CMP.B  #5,D4            \ IS THE TRANSFER OVERLAP ?
00000430 6C08               BGE.S  G_DMA_W2         /
```



```
00000432 0C2B 00FF G_DMA_W1 CMPI.B #255,T_SC_OFF(A3)  IF NOT THEN WAIT TILL DONE
                 0005
00000438 66F8               BNE.S  G_DMA_W1
0000043A 4E75       G_DMA_W2 RTS
                        *
                        *
                        *
                        *
                        *
0000043C 49FA FBE0 GD_STBSY LEA    EXTG_EG_TDMA,A4  \ save g_dmaterm routine
00000440 4EB9 0000          JSR    DMA_STBSY        / in dma temps          475 TM 9/17/82
                 0000
00000446 4E75               RTS                                             475 TM 9/17/82
```

```
                        *
                        *          Transfer INTERRUPT
                        *
00000448 177C 0001 G_T_INT   MOVE.B #TT_INT,TACT_OFF(A3)   set tfr type to INTERRUPT
              0007
0000044E 6006                BRA.S  G_T_BIC                go to common code
                        *
                        *          Transfer BURST ( intr on 1st byte FHS on rest )
                        *
00000450 177C 0003 G_T_BST   MOVE.B #TT_BURST,TACT_OFF(A3) set tfr type to BURST
              0007
                        *          BRA.S  G_T_BIC                go to common code
                        *
                        *          common interrupt and burst code
                        *
00000456 4EB9 0000 G_T_BIC   JSR    STBSY                 SET BUFFER BUSY, ETC
              0000
0000045C 4A2B 000D           TST.B  TDIR_OFF(A3)          \ test for transfer direction
00000460 6E06                BGT.S  G_TOI                 /
                        *
                        *          Transfer Input Interrupt or Transfer Input Burst
                        *
              0000 0462 G_TII     EQU    *
00000462 4A69 0004 G_INT_I   TST    4(A1)                 Dummy read to set I/O line to I
00000466 1280                MOVE.B D0,(A1)               wti 7,0 (trigger input)
                        *
                        *          Transfer Output Interrupt or Transfer Output Burst
                        *
              0000 0468 G_TOI     EQU    *
00000468 137C 0003 G_INT_O   MOVE.B #3,2(A1)              Allow I/O to cause an interrupt
              0002
0000046E 102A 002C           MOVE.B EIRB_OFF(A2),D0       GET CTL0/1
00000472 0000 0080           ORI.B  #$80,D0
00000476 6100 FEB8           BSR    G_WII5                Enable interrupts
0000047A 60AC                BRA    G_DMA_W               and wait if necessary
```



```
                        *
                        *          Transfer FHS
                        *
0000047C 177C 0004 G_T_FHS   MOVE.B #TT_FHS,TACT_OFF(A3)   set tfr type to FHS
              0007
00000482 4EB9 0000           JSR    STBSY                 set buffer busy         475 TM 9/17/82
              0000
00000488 4EB9 0000           JSR    ITXFR                 get all pointers        475 TM 9/17/82
              0000
0000048E 4A2B 000D           TST.B  TDIR_OFF(A3)          \ test for transfer direction
00000492 6E54                BGT.S  G_TFO                 /
                        *
                        *          Transfer FHS in
                        *
00000494 4A69 0004 G_TFI     TST    4(A1)                 to set I/O line to I    507 TM 9/17/82
00000498 1280                MOVE.B D0,(A1)               trigger input           507 TM 9/17/82
0000049A 7000                MOVEQ  #0,D0                 clear upper part  of data in
0000049C 206B 0020           MOVEA.L TFIL_OFF(A3),A0      GET FILL POINTER
000004A0 4A2B 000A           TST.B  T_BW_OFF(A3)          GET B/W INDICATIONS
000004A4 661C                BNE.S  G_TFIW
000004A6 6100 FCE6 G_TFIB    BSR    G_WAIT
000004AA 1029 0005 G_TFIB1   MOVE.B 5(A1),D0              D0.L = Byte received
000004AE 10C0                MOVE.B D0,(A0)+
000004B0 5383                SUBQ.L #1,D3                 Decrement transfer counter
000004B2 6F24                BLE.S  GTFI_TRM              If buffer full; exit fast handshake
000004B4 B440                CMP.W  D0,D2                 If input character matches end character
000004B6 6720                BEQ.S  GTFI_TRM                then we're done
000004B8 1280                MOVE.B D0,(A1)               wti 7,0 (Trigger next input)
000004BA 0811 0000 G_TFIB2   BTST   #0,(A1)               PFLG = 1 ?
000004BE 66EA                BNE    G_TFIB1               Yes: Get next byte
000004C0 60F8                BRA    G_TFIB2               No:  Keep checking
000004C2 6100 FCCA G_TFIW    BSR    G_WAIT
000004C6 30E9 0004 G_TFIW1   MOVE   4(A1),(A0)+          Copy word from GPIO to buffer
000004CA 5383                SUBQ.L #1,D3                 Decrement transfer counter
000004CC 6F0A                BLE.S  GTFI_TRM              If D3 = 0 we're done
000004CE 1280                MOVE.B D0,(A1)               wti 7,0 (Trigger next input)
000004D0 0811 0000 G_TFIW2   BTST   #0,(A1)               PFLG = 1
000004D4 66F0                BNE    G_TFIW1
000004D6 60F8                BRA    G_TFIW2               No:  Keep checking
                        *
                        * FHS TRANSFER TERMINATION
                        *
000004D8 2743 0010 GTFI_TRM MOVE.L  D3,TCNT_OFF(A3)       D3 has bytes not finished
000004DC 2748 0020          MOVE.L  A0,TFIL_OFF(A3)       update fill pointer
000004E0 4EB9 0000 GTFO_TRM JSR     STCLR                 MARK BUFFER FINISHED    475 TM 9/17/82
              0000
000004E6 4E75               RTS
                        *
```

```
1038                            *         Transfer FHS out
1039                            *
1040    000004E8 206B 001C G_TFO    MOVEA.L TEMP_OFF(A3),A0         GET EMPTY POINTER
1041    000004EC 4A2B 000A          TST.B   T_BW_OFF(A3)            GET B/W INDICATIONS
1042    000004F0 6616               BNE.S   G_TFOW
1043    000004F2 6100 FC9A G_TFO8   BSR     G_WAIT                  Wait for card ready
1044    000004F6 1358 0005 G_TFOB1  MOVE.B  (A0)+,5(A1)             Copy byte from buffer to GPIO
1045    000004FA 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger next output)
1046    000004FC 5383               SUBQ.L  #1,D3                   Decrement transfer count
1047    000004FE 6F1E               BLE.S   G_TFO3                  If D3 = 0 then we're done
1048    00000500 0811 0000 G_TFOB2  BTST    #0,(A1)                 PFLG = 1 ?
1049    00000504 66F0               BNE     G_TFOB1                 Yes: Get next byte
1050    00000506 60F8               BRA     G_TFOB2                 No:  Keep checking
1051
1051
1051
1052    00000508 6100 FC84 G_TFOW   BSR     G_WAIT                  wait for card ready
1053    0000050C 3358 0004 G_TFOW1  MOVE.W  (A0)+,4(A1)             Copy word from buffer to GPIO
1054    00000510 1280               MOVE.B  D0,(A1)                 wti 7,0 (Trigger next output)
1055    00000512 5383               SUBQ.L  #1,D3                   Decrement transfer count
1056    00000514 6F08               BLE.S   G_TFO3                  IF D3 = 0 then we're done
1057    00000516 0811 0000 G_TFOW2  BTST    #0,(A1)                 PFLG = 1 ?
1058    0000051A 66F0               BNE     G_TFOW1                 Yes: Get next byte
1059    0000051C 60F8               BRA     G_TFOW2                 No:  Keep checking
1060
1060
1060
1061    0000051E 2748 001C G_TFO3   MOVE.L  A0,TEMP_OFF(A3)         SAVE EMPTY PTR
1062    00000522 42AB 0010          CLR.L   TCNT_OFF(A3)            CLEAR COUNT
1063    00000526 60B8               BRA.S   GTFO_TRM
1064
1064
1064
1064
```



```
1066                            ******************************************************
1067                            *
1068                            *  UNDMA
1069                            *
1070                            *  Release the DMA channel associated with the GPIO
1071                            *  card when the transfer is done.
1072                            *
1073                            ******************************************************
1074            0000 0528 G_DMATERM     EQU *
1075    00000528 007C 2700 G_UNDMA  ORI     #$2700,SR               Disable all other interrupts
1076    0000052C 4EB9 0000          JSR     DROPDMA                 Release the DMA channel 475 TM 9/17/82
                 0000
1077    00000532 4EB9 0000          JSR     ITXFR                   See if buf was active   475 TM 9/17/82
                 0000
1078    00000538 6756               BEQ.S   G_UNDEND                No: this should never happen
1079    0000053A 4A2B 000D          TST.B   TDIR_OFF(A3)            Was it an Input (0) or Output (1) ?
1080    0000053E 672A               BEQ.S   UNINPUT
1081    00000540 102A 002C UNOUTPUT MOVE.B  EIRB_OFF(A2),D0
1082    00000544 C07C 000F          AND     #$F,D0
1083    00000548 6100 FDE4          BSR     G_WTI5
1084    0000054C 337C 0300          MOVE.W  #$300,2(A1)             Restore REDYN and EIREN
                 0002
1085    00000552 4A2B 000A          TST.B   T_BW_OFF(A3)            Was it a Byte (1) or Word (0) tfr ?
1086    00000556 6702               BEQ.S   UNOUT2
1087    00000558 D083               ADD.L   D3,D3                   Word: Double the count
1088    0000055A D7AB 001C UNOUT2   ADD.L   D3,TEMP_OFF(A3)         UPDATE EMPTY POINTER
1089    0000055E 42AB 0010          CLR.L   TCNT_OFF(A3)            CLEAR COUNT
1090    00000562 4EB9 0000          JSR     STCLR                   Unbusy the buffer       475 TM 9/17/82
                 0000
1091    00000568 4E75               RTS
1092
1092
1092
1092
1093    0000056A 202B 0010 UNINPUT  MOVE.L  TCNT_OFF(A3),D0         D0 = Transfer count
1094                            * tm    MOVE.L  D4,TCNT_OFF(A3)     SET COUNT TO REMAINING BYTES
1095    0000056E 9084               SUB.L   D4,D0                   GET ACTUAL BYTES TFR'D
1096    00000570 4A2B 000A          TST.B   T_BW_OFF(A3)            Was it a Byte (1) or Word (0) tfr ?
1097    00000574 6702               BEQ.S   UNIN2
1098    00000576 D080               ADD.L   D0,D0                   Word: Double the count
1099    00000578 D1AB 0020 UNIN2    ADD.L   D0,TFIL_OFF(A3)         UPDATE FILL POINTER
1100    0000057C 277C 0000          MOVE.L  #1,TCNT_OFF(A3)         Last byte is received via interrupt
                 0001 0010
1101    00000584 337C 0380          MOVE.W  #$380,2(A1)             Allow it to cause an interrupt
                 0002
1102    0000058A 177C 0001          MOVE.B  #TT_INT,TACT_OFF(A3)    change tfr type
                 0097
1103    00000590 4E75       G_UNDEND RTS
```

```
PAGE  31 [3.0] 12/26/84 21:31:49 IOLIB EXTG - INTERRUPT SERVICE ROUTINE                    *** File name: GPIO ***

1106                        ******************************************************
1107                        *
1108                        *  GPIO INTERRUPT SERVICE ROUTINE
1109                        *
1110                        ******************************************************
1111              0000 0592 G_ISR     EQU     *
1112    00000592 4EB9 0000 G_ISR1    JSR     ITXFR                   Transfer in progress?   475 TM 9/17/82
                  0000
1113    00000598 6614                BNE.S   G_ATFR
1114    0000059A 102A 002C G_ISR2    MOVE.B  EIRB_OFF(A2),D0         NO TFR IN PROGRESS
1115    0000059E C07C 000F           AND     #$F,D0
1116    000005A2 6100 FD8A           BSR     G_WTIS
1117    000005A6 4EB9 0000           JSR     LOGINT                  log the interrupt       475 TM 9/17/82
                  0000
1118    000005AC 4E75       GISR_END RTS
1119
1119
1119
1119
1120    000005AE 1029 0003 G_ATFR    MOVE.B  3(A1),D0                Put DMA channel for GPIO in D0
1121    000005B2 C07C 0003           AND     #3,D0
1122    000005B6 6648                BNE.S   G_TABORT                If DMA in progress: Abort & update pointers
1123    000005B8 B23C 0004           CMP.B   #TT_FHS,D1              \  If FHS then exit
1124    000005BC 6742                BEQ.S   G_TABORT                /
1125    000005BE 923C 0001 G_ATFR1   SUB.B   #1,D1                   \ IF INTR THEN D1=0
1126    000005C2 E509                LSL.B   #2,D1                   /    BURST THEN D1=8
1127    000005C4 4A2B 000A           TST.B   T_BW_OFF(A3)            \ IF WORD THEN ADD 4
1128    000005C8 6704                BEQ.S   GTST_DIR                /
1129    000005CA D23C 0004           ADD.B   #4,D1
1130    000005CE 4A2B 000D GTST_DIR  TST.B   TDIR_OFF(A3)            \ IF OUTPUT THEN ADD 16
1131    000005D2 6708                BEQ.S   G_ATFR2                 /
1132    000005D4 D23C 0010           ADD.B   #$10,D1
1133    000005D8 4881                EXT.W   D1
1134    000005DA 48C1                EXT.L   D1
1135    000005DC 4EFB 1002 G_ATFR2   JMP     G_TTBL(D1)              computed goto
1136
1136
1136
1136
1137    000005E0 6000 00C0 G_TTBL    BRA     G_TIIB                  INTR IN BYTE
1138    000005E4 6000 00D0           BRA     G_TIIW                  INTR IN WORD
1139    000005E8 6000 0084           BRA     G_TIFB                  BRST IN BYTE
1140    000005EC 6000 009E           BRA     G_TIFW                  BRST IN WORD
1141    000005F0 6000 0044           BRA     G_TOIB                  INTR OUT BYTE
1142    000005F4 6000 0046           BRA     G_TOIW                  INTR OUT WORD
1143    000005F8 6000 0010           BRA     G_TOFB                  BRST OUT BYTE
1144    000005FC 6000 0022           BRA     G_TOFW                  BRST OUT WORD
1145
1145
1145
1145
1146    00000600 4EB9 0000 G_TABORT  JSR     DROPDMA                 Release DMA channel     475 TM 9/17/82
                  0000
1147    00000606 6000 00C2           BRA     G_TDIN
```

```
PAGE  32 [3.0] 12/26/84 21:31:49 IOLIB EXTG - INTERRUPT SERVICE ROUTINE                    *** File name: GPIO ***

1149                        ************************************************
1150                        *
1151                        *       OUTPUT TRANSFERS
1152                        *
1153                        ************************************************
1154    0000060A 007C 2700 G_TOFB    ORI     #$2700,SR               Disable interrupts
1155    0000060E 1358 0005 G_TOFB1   MOVE.B  (A0)+,5(A1)             Copy byte from buffer to GPIO
1156    00000612 5383                SUBQ.L  #1,D3                   Decrement transfer count
1157    00000614 6F3A                BLE.S   G_TDOUT                 If D3 = 0 then we're done
1158    00000616 1280                MOVE.B  D0,(A1)                 wti 7,0 (Trigger next output)
1159    00000618 0811 0000 G_TOFB2   BTST    #0,(A1)                 PFLG = 1 ?
1160    0000061C 66F0                BNE     G_TOFB1                 Yes: Get next byte
1161    0000061E 60F8                BRA     G_TOFB2                 No:  Keep checking
1162
1162
1162
1163    00000620 007C 2700 G_TOFW    ORI     #$2700,SR               Disable interrupts
1164    00000624 3358 0004 G_TOFW1   MOVE.W  (A0)+,4(A1)             Copy word from buffer to GPIO
1165    00000628 5383                SUBQ.L  #1,D3                   Decrement transfer count
1166    0000062A 6F24                BLE.S   G_TDOUT                 IF D3 = 0 then we're done
1167    0000062C 1280                MOVE.B  D0,(A1)                 wti 7,0 (Trigger next output)
1168    0000062E 0811 0000 G_TOFW2   BTST    #0,(A1)                 PFLG = 1 ?
1169    00000632 66F0                BNE     G_TOFW1                 Yes: Get next byte
1170    00000634 60F8                BRA     G_TOFW2                 No:  Keep checking
1171
1171
1171
1171
1172    00000636 1358 0005 G_TOIB    MOVE.B  (A0)+,5(A1)             Copy a byte from buffer to GPIO
1173    0000063A 6004                BRA.S   G_ENDOUT
1174    0000063C 3358 0004 G_TOIW    MOVE.W  (A0)+,4(A1)             Copy a word from buffer to GPIO
1175    00000640 2748 001C G_ENDOUT  MOVE.L  A0,TEMP_OFF(A3)         Save A0 for use on next output byte/word
1176    00000644 5383                SUBQ.L  #1,D3
1177    00000646 2743 0010           MOVE.L  D3,TCNT_OFF(A3)         Decrement transfer out counter
1178    0000064A 6F04                BLE.S   G_TDOUT                 If zero, go thru Transfer Done Out
1179    0000064C 1280                MOVE.B  D0,(A1)                 wti 7,0 (Trigger byte out)
1180    0000064E 4E75                RTS                             End of ISR
1181
1181
1181
1181
1182    00000650 2748 001C G_TDOUT   MOVE.L  A0,TEMP_OFF(A3)         save empty ptr
1183    00000654 42AB 0010           CLR.L   TCNT_OFF(A3)            clear tfr count
1184    00000658 137C 0000           MOVE.B  #0,3(A1)                Stop card from interrupting
                  0003
1185    0000065E 136A 002C           MOVE.B  EIRB_OFF(A2),7(A1)      Put EIR byte in CTL1/CTL0
                  0007
1186    00000664 1280                MOVE.B  D0,(A1)                 wti 7,0 (Trigger last output)
1187    00000666 4EB9 0000           JSR     STCLR                   Unbusy buffer           475 TM 9/17/82
                  0000
1188    0000066C 4E75                RTS
```

```
1190                        ********************************************
1191                        *
1192                        *       INPUT TRANSFERS
1193                        *
1194                        ********************************************
1195    0000066E 007C 2700 G_TIFB   ORI     #$2700,SR
1196    00000672 7000               MOVEQ   #0,D0
1197    00000674 1029 0005 G_TIFB1  MOVE.B  5(A1),D0            D0.L = Byte received
1198    00000678 10C0               MOVE.B  D0,(A0)+
1199    0000067A 5383               SUBQ.L  #1,D3               Decrement transfer counter
1200    0000067C 6F4C               BLE.S   G_TDIN              If buffer full; exit fast handshake
1201    0000067E B440               CMP.W   D0,D2               If input character matches end character
1202    00000680 6748               BEQ.S   G_TDIN                then we're done
1203    00000682 1280               MOVE.B  D0,(A1)             wti 7,0 (Trigger next input)
1204    00000684 0811 0000 G_TIFB2  BTST    #0,(A1)             PFLG = 1 ?
1205    00000688 66EA               BNE     G_TIFB1             Yes: Get next byte
1206    0000068A 60F8               BRA     G_TIFB2             No:  Keep checking
1207
1207
1207
1207
1208    0000068C 007C 2700 G_TIFW   ORI     #$2700,SR           Disable interrupts
1209    00000690 30E9 0004 G_TIFW1  MOVE    4(A1),(A0)+         Copy word from GPIO to buffer
1210    00000694 5383               SUBQ.L  #1,D3               Decrement transfer counter
1211    00000696 6F32               BLE.S   G_TDIN              If D3 = 0 we're done
1212    00000698 1280               MOVE.B  D0,(A1)             wti 7,0 (Trigger next input)
1213    0000069A 0811 0000 G_TIFW2  BTST    #0,(A1)             PFLG = 1
1214    0000069E 66F0               BNE     G_TIFW1
1215    000006A0 60F8               BRA     G_TIFW2             No:  Keep checking
1216
1216
1216
1216
1217    000006A2 7000       G_TIIB  MOVEQ   #0,D0
1218    000006A4 1029 0005          MOVE.B  5(A1),D0            D0.L = Byte received
1219    000006A8 10C0               MOVE.B  D0,(A0)+            Store byte in input buffer
1220    000006AA B440               CMP.W   D0,D2               Compare termination byte with input byte
1221    000006AC 660C               BNE.S   G_ENDIN             No match: Everybody get out of here !
1222    000006AE 5383               SUBQ.L  #1,D3               Decrement byte counter
1223    000006B0 2743 0010          MOVE.L  D3,TCNT_OFF(A3)
1224    000006B4 6014               BRA.S   G_TDIN
1225
1225
1225
1225
1226    000006B6 30E9 0004 G_TIIW   MOVE    4(A1),(A0)+         Copy word from GPIO to buffer
1227    000006BA 2748 0020 G_ENDIN  MOVE.L  A0,TFIL_OFF(A3)     Save buffer pointer
1228    000006BE 5383               SUBQ.L  #1,D3
1229    000006C0 2743 0010          MOVE.L  D3,TCNT_OFF(A3)
1230    000006C4 6F04               BLE.S   G_TDIN
1231    000006C6 1230               MOVE.B  D0,(A1)             wti 7,0
1232    000006C8 4E75               RTS
1233
1233
1233
1233
1234    000006CA 2748 0020 G_TDIN   MOVE.L  A0,TFIL_OFF(A3)     save fill ptr
```



```
1235    000006CE 2743 0010          MOVE.L  D3,TCNT_OFF(A3)     save remaining count
1236    000006D2 137C 0000          MOVE.B  #0,3(A1)            Stop card from interrupting
                 0003
1237    000006D8 136A 002C          MOVE.B  EIRB_OFF(A2),7(A1)  Put EIR byte in CTL1/CTL0
                 0007
1238    000006DE 4EB9 0000          JSR     STCLR               Unbusy buffer          475 TM 9/17/82
                 0000
1239    000006E4 4E75               RTS
```

```
 1241                                   END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```


*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ABORT_IO	ABS	148	00000008
CHECK_TFR	ABS	155	00000019
CHECK_TIMER	ABS	156	0000001C
DELAY_TIMER	ABS	157	0000001F
DMA_STBSY	ABS	153	00000013
DROPDMA	ABS	151	0000000F
GETDMA	ABS	150	0000000D
IODECLARATIONS	ABS	312	00000022
ITXFR	ABS	147	00000006
LOGINT	ABS	149	0000000B
STBSY	ABS	145	00000002
STCLR	ABS	146	00000004
SYSGLOBALS	ABS	313	00000026
TESTDMA	ABS	152	00000011
WAIT_TFR	ABS	154	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	252			00000034
BADTMO	ABS	330			0000000B
BAD_RDS	ABS	338			00000013
BAD_SCT	ABS	339			00000014
BUF1_OFF	ABS	243			00000024
BUF0_OFF	ABS	244			00000028
BUF_BUSY	ABS	328			00000009
CCR	STREG	0			00000005
CRD_DOWN	REL	636			000001EE
CRD_DWN	ABS	340			00000015
C_ADR	ABS	242			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
DFC	STREG	0			00000008
EIRB_OFF	ABS	245			0000002C
EOD_SEEN	ABS	341			00000016
ESC_CODE	ABS	349	SYSGLOBALS	+	FFFFFFFE
ESC_ERR	REL	645			000001FC

EXTG_EG_CLR	REL	485	000000BC
EXTG_EG_INIT	REL	370	00000002
EXTG_EG_ISR	REL	379	00000010
EXTG_EG_RDB	REL	397	0000002C
EXTG_EG_RDS	REL	441	00000074
EXTG_EG_RDW	REL	419	00000050
EXTG_EG_SET	REL	475	000000AC
EXTG_EG_TDMA	REL	388	0000001E
EXTG_EG_TEST	REL	495	000000CC
EXTG_EG_TFR	REL	465	0000009C
EXTG_EG_WTB	REL	409	00000040
EXTG_EG_WTC	REL	454	0000008A
EXTG_EG_WTW	REL	431	00000064
EXTG_EXTG	REL	365	00000000
GD_STBSY	REL	953	0000043C
GISR_END	REL	1118	000005AC
GREGEXIT	REL	771	000002B8
GR_CARD	REL	769	000002B2
GR_CARD1	REL	770	000002B4
GSET_1	REL	682	00000230
GTERR_8	REL	640	000001F6
GTERR_D	REL	642	000001FA
GTFI_TRM	REL	1032	000004D8
GTFO_TRM	REL	1034	000004E0
GTSTI	REL	711	00000266
GTST2	REL	716	00000270
GTST3	REL	719	000C027A
GTST4	REL	722	00000280
GTST_CHK	REL	712	0000026A
GTST_DIR	REL	1130	000005CE
GTST_RTS	REL	714	0000026E
GTST_SET	REL	713	0000026C
GWTCTBL	REL	813	000002F2
G_ATFR	REL	1120	000005AE
G_ATFR1	REL	1125	000005BE
G_ATFR2	REL	1135	000005DC
G_CLR	REL	690	00000236
G_CRDREG	ABS	736	00000000
G_DATA_0	REL	830	0000031C
G_DMA1	REL	917	000003DA
G_DMA3	REL	925	000003F6
G_DMA4	REL	936	0000040E
G_DMA6	REL	942	00000424
G_DMAOUT	REL	932	000003FC
G_DMATERM	REL	1074	00000528
G_DMA_W	REL	946	00000428
G_DMA_W1	REL	949	00000432
G_DMA_W2	REL	951	0000043A
G_EIR	REL	838	00000322
G_ENDIN	REL	1227	000006BA
G_ENDOUT	REL	1175	00000640
G_INIT	REL	512	000000E2
G_INT_I	REL	979	00000462
G_INT_O	REL	985	00000468
G_IOFS	REL	526	00000106
G_IOFS1	REL	530	00000112
G_IOFS2	REL	533	0000011A

G_ISR	REL	1111	00000592
G_ISR1	REL	1112	00000592
G_ISR2	REL	1114	0000059A
G_NODMA	REL	869	0000037E
G_RDB	REL	551	00000128
G_RDS	REL	739	00000288
G_RDSTBL	REL	788	000002D8
G_RDS_DI	REL	777	000002BE
G_RDS_PST	REL	784	000002CE
G_RDS_RDY	REL	780	000002C4
G_RESET	REL	513	000000E2
G_ROUTINE	ABS	734	00000002
G_RTS	REL	847	00000344
G_RTS1	REL	848	00000346
G_RT_END	REL	795	000002E4
G_RT_SIZ	ABS	796	0000000C
G_SCBSY	REL	634	000001EA
G_SC_ERR	REL	638	000001F2
G_SET	REL	673	00000212
G_SET_CTL	REL	825	00000308
G_SET_PCT	REL	822	00000302
G_STSO	REL	565	00000150
G_STSOB	REL	571	00000164
G_STSOC	REL	573	0000016E
G_STS1	REL	569	0000015C
G_STS1B	REL	580	0000018A
G_STSCHK	REL	559	0000013A
G_STSRTS	REL	568	0000015A
G_TABORT	REL	1146	00000600
G_TBL	REL	876	00000388
G_TDIN	REL	1234	000006CA
G_TDOUT	REL	1182	00000650
G_TEMP	ABS	735	00000001
G_TEST	REL	705	00000254
G_TFI	REL	1002	00000494
G_TFIB	REL	1008	000004A6
G_TFIB1	REL	1009	000004AA
G_TFIB2	REL	1016	000004BA
G_TFIW	REL	1020	000004C2
G_TFIW1	REL	1021	000004C6
G_TFIW2	REL	1025	000004D0
G_TFO	REL	1040	000004E8
G_TFO3	REL	1061	0000051E
G_TFOB	REL	1043	000004F2
G_TFOB1	REL	1044	000004F6
G_TFOB2	REL	1048	00000500
G_TFOW	REL	1052	00000508
G_TFOW1	REL	1053	0000050C
G_TFOW2	REL	1057	00000516
G_TFR	REL	857	0000034E
G_TID	REL	911	000003C6
G_TIFB	REL	1195	0000066E
G_TIFB1	REL	1197	00000674
G_TIFB2	REL	1204	00000684
G_TIFW	REL	1208	0000068C
G_TIFW1	REL	1209	00000690
G_TIFW2	REL	1213	0000069A

```
G_TII           REL      978                        00000462
G_TIIB          REL     1217                        000006A2
G_TIIW          REL     1226                        000006B6
G_TOD           REL      931                        000003FC
G_TOFB          REL     1154                        0000060A
G_TOFB1         REL     1155                        0000060E
G_TOFB2         REL     1159                        00000618
G_TOFW          REL     1163                        00000620
G_TOFW1         REL     1164                        00000624
G_TOFW2         REL     1168                        0000062E
G_TOI           REL      984                        00000468
G_TOIB          REL     1172                        00000636
G_TOIW          REL     1174                        0000063C
G_TTBL          REL     1137                        000005E0
G_T_BIC         REL      972                        00000456
G_T_BST         REL      966                        00000450
G_T_DMA         REL      903                        000003B0
G_T_FHS         REL      994                        0000047C
G_T_INT         REL      980                        00000448
G_UNDEND        REL     1103                        00000590
G_UNDMA         REL     1075                        00000528
G_WAIT          REL      589                        0000018E
G_WAIT2         REL      595                        000001A2
G_WAIT2B        REL      612                        000001C0
G_WAIT3         REL      601                        000001AA
G_WAIT3B        REL      614                        000001C6
G_WAIT4         REL      608                        000001B8
G_WAIT4B        REL      615                        000001CC
G_WAIT5B        REL      624                        000001E6
G_WAITER        REL      605                        000001B4
G_WRTS          REL      610                        000001BE
G_WTB           REL      540                        0000011C
G_WTC           REL      807                        000002E4
G_WTC_RST       REL      820                        000002FE
G_WTI5          REL      841                        0000032E
H_ISR_PM        ABS      241                        000001C
H_ISR_PR        ABS      239                        00000014
H_ISR_SL        ABS      240                        00000018
IOE_ERROR       ABS      344                        FFFFFFE6
IOE_RSLT        ABS      346 IODECLARATIONS  +      FFFFFFBE
IOE_SC          ABS      347 IODECLARATIONS  +      FFFFFFBA
IO_MISC         ABS      342                        00000017
IO_SC           ABS      246                        0000002D
ISR_ENTRY       ABS      237                        00000000
MA              ABS      251                        00000033
MA_W            ABS      250                        00000032
NOT_HPIB        ABS      321                        00000002
NOT_LSTN        ABS      335                        00000010
NOT_TALK        ABS      334                        0000000F
NO_ACTL         ABS      322                        00000003
NO_CARD         ABS      320                        00000001
NO_DATA         ABS      325                        00000006
NO_DMA          ABS      332                        0000000D
NO_DRV          ABS      331                        0000000C
NO_DVC          ABS      323                        00000004
NO_SCTL         ABS      337                        00000012
NO_SPACE        ABS      324                        00000005
```



```
NO_WORD         ABS      333                        0000000E
RCVR_BLK        ABS      350 SYSGLOBALS      +      FFFFFFF6
RDS_ERR         REL      756                        000002AC
SC_BUSY         ABS      327                        00000008
SFC             STREG      0                        00000009
SP              AREG       0                        00000007
SR              STREG      0                        00000006
SYSFLAG2        ABS      353                        FFFFFEDA
TACT_OFF        ABS      267                        00000007
TBSZ_OFF        ABS      290                        00000018
TBUF_OFF        ABS      289                        00000014
TCHR_OFF        ABS      286                        0000000E
TCNTERR         ABS      329                        0000000A
TCNT_OFF        ABS      288                        00000010
TDIR_OFF        ABS      284                        0000000D
TEMP_OFF        ABS      291                        0000001C
TEND_OFF        ABS      282                        0000000B
TFIL_OFF        ABS      292                        00000020
TFR_ERR         ABS      326                        00000007
TIMEOUT         ABS      247                        0000002E
TIMER_PRESENT   ABS      352                        00000001
TMO_ERR         ABS      336                        00000011
TTMP_OFF        ABS      265                        00000000
TT_BURST        ABS      303                        00000003
TT_DMA          ABS      302                        00000002
TT_FHS          ABS      304                        00000004
TT_INT          ABS      301                        00000001
TUSR_OFF        ABS      268                        00000009
T_BW_OFF        ABS      280                        0000000A
T_DMAPRI        ABS      297                        00000030
T_PM_OFF        ABS      296                        0000002C
T_PR_OFF        ABS      293                        00000024
T_SC_OFF        ABS      266                        00000005
T_SL_OFF        ABS      295                        00000028
UNIN2           REL     1099                        00000578
UNINPUT         REL     1093                        0000056A
UNOUT2          REL     1088                        0000055A
UNOUTPUT        REL     1081                        00000540
USER_ISR        ABS      238                        00000014
USP             STREG      0                        00000007
VBR             STREG      0                        0000000A
```

Description

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

```
PAGE   1 [3.0] 12/26/84 21:38:26  ASSEMBLY OF GPIODVR.TEXT                                   *** File name: GPIODVR ***
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)  setpctl: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}
```

```
PAGE   2 [3.0] 12/26/84 21:38:26  ASSEMBLY OF GPIODVR.TEXT                                   *** File name: GPIODVR ***
   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                                   refa    check_timer                 jws 8/10/83
   87
   88                                   lmode   mini_ioresc
   89                                   lmode   check_timer                 jws 8/10/83
   90
   91              0000 0001 timer_present  equ 1                           jws 8/10/83
   92              FFFF FEDA sysflag2       equ $fffffeda                   jws 8/10/83
   93
   94
   95
   96                           *
   97                           * module initialization routine
   98                           *
   99   00000000 4E75           gp_gp           rts
  100
  101
  102                          *
  103                          * ioresult assignments
  104                          *
  105             0000 0004 ztimeout       equ     4
  106             0000 0015 zcatchall      equ     21
  107
  108
  109                          *
  110                          * error exits
  111                          *
  112   00000002 7015          ioresc_catchall moveq   #zcatchall,d0       zcatchall ioresult
  113
  114   00000004 3F00          ioresc          move    d0,-(sp)            push the ioresult
  115   00000006 4EB9 0000             jsr     mini_ioresc                 set ioresult then escape(-10)
                 0000
```

```
PAGE   3 [3.0] 12/26/84 21:38:26  ASSEMBLY OF GPIODVR.TEXT                                  *** File name: GPIODVR ***

  116
  117
  118    0000000C 3B7C FFF4 bus_error       move    #-12,sysglobals-2(a5)   set the escapecode
                  FFFE
  119    00000012 4E4A                      trap    #10                     escape
```

```
PAGE   4 [3.0] 12/26/84 21:38:26  ASSEMBLY OF GPIODVR.TEXT                                  *** File name: GPIODVR ***

  121                              *
  122                              * gpiowaitready with 2 second timeout
  123                              *
  124    00000014 203C 0000 waitready       move.l  #206,d0              timeout counter
                  00CE
  125                              *
  126                              * Counter changed to be about 1 ms at 16 MHz -- was 206185
  127                              *   by jws 8/10/83
  128                              *
  129
  130    0000001A 0829 0003 waitready_loop  btst    #3,7(a1)                peripheral status?
                  0007
  131    00000020 66E0                      bne     ioresc_catchall         ioresc(zcatchall) if so
  132    00000022 0811 0000                 btst    #0,(a1)                 ready?
  133    00000026 6650                      bne.s   waitready_rts           branch if so
  134    00000028 5380                      subq.l  #1,d0                   decrement the timeout counter
  135    0000002A 6EEE                      bgt     waitready_loop          loop until timeout count expired
  136                              *
  137                              *   Low data rate if we get here, so use timer if have it    jws 8/10/83
  138                              *   8 MHz loop code is duplicated from above                 jws 8/10/83
  139                              *                                                            jws 8/10/83
  140    0000002C 0838 0001                 btst    #timer_present,sysflag2                  jws 8/10/83
                  FEDA
  141    00000032 671A                      beq.s   waitready_timer         use timer    jws 8/10/83
  142    00000034 203C 0003                 move.l  #206185,d0       else set 2 sec loop jws 8/10/83
                  2569
  143    0000003A 0829 0003 waitready_loop2  btst    #3,7(a1)                 peripheral status?
                  0007
  144    00000040 66C0                      bne     ioresc_catchall         ioresc(zcatchall) if so
  145    00000042 0811 0000                 btst    #0,(a1)                 ready?
  146    00000046 6630                      bne.s   waitready_rts           branch if so
  147    00000048 5380                      subq.l  #1,d0                   decrement the timeout counter
  148    0000004A 6EEE                      bgt     waitready_loop2         loop until timeout count expired
  149    0000004C 602C                      bra.s   wait_timeout            timeout on 2sec loop  jws
  150                              *
  151              0000 004E waitready_timer equ    *                                        jws 8/10/83
  152    0000004E 1F3C 0001                 move.b  #1,-(sp)          setup timer record  jws 8/10/83
  153    00000052 2F3C 0000                 move.l  #2000,-(sp)       time to wait        jws 8/10/83
                  07D0
  154    00000058 0829 0003 waitready_tloop btst    #3,7(a1)          check psts          jws 8/10/83
                  0007
  155    0000005E 66A2                      bne     ioresc_catchall   if not there,escape jws 8/10/83
  156    00000060 0811 0000                 btst    #0,(a1)           check card ready    jws 8/10/83
  157    00000064 6610                      bne.s   waitready_texit  if so then done     jws 8/10/83
  158    00000066 4857                      pea     (sp)              point to timer rec  jws 8/10/83
  159    00000068 4EB9 0000                 jsr     check_timer       timed out?          jws 8/10/83
                  0000
  160    0000006E 6AE8                      bpl     waitready_tloop   no--try again       jws 8/10/83
  161    00000070 5C4F                      addq    #6,sp             yes, clean stack    jws 5/2/84
  162    00000072 700A                      moveq   #10,d0            give one more try   jws 5/2/84
  163    00000074 60C4                      bra     waitready_loop2   with a short count  jws 5/2/84
  164    00000076 5C4F      waitready_texit addq    #8,sp             cleanup stack       jws 8/10/83
  165    00000078 4E75      waitready_rts   rts                       and continue        jws 8/10/83
  166
  167
  168    0000007A 7004      wait_timeout    moveq   #ztimeout,d0              ztimeout ioresult
  169    0000007C 6086                      bra     ioresc                    escape
```

```
                        ********************************************************************************
                        *                                  gpioclear                                   *
                        ********************************************************************************

0000007E 205F           gp_gpioclear    movea.l (sp)+,a0                pop the return address
00000080 225F                           movea.l (sp)+,a1                pop the gpio card base address
00000082 1340 0001                      move.b  d0,1(a1)                reset the card
00000086 203C 0000                      move.l  #200,d0                 prepare to...
         00C8
                        *
                        * Count changed from 100 to 200 for 16 MHz processors jws 8/10/83
                        *
0000008C 51C8 FFFE                      dbra    d0,*                    wait a while...
00000090 4229 0007                      clr.b   7(a1)                   clear ctl1 & clt0
00000094 6100 FF7E                      bsr     waitready               before testing psts & ready
00000098 4ED0                           jmp     (a0)                    return


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

0000009A 205F           gp_gpiowordout  movea.l (sp)+,a0                pop the return address
0000009C 226F 0002                      movea.l 2(sp),a1                gpio card base address
000000A0 6100 FF72                      bsr     waitready               wait until ready
000000A4 3357 0004                      move    (sp),4(a1)              output the datum
000000A8 1280                           move.b  d0,(a1)                 set pctl
000000AA 5C8F                           addq.l  #6,sp                   pop off the parameters
000000AC 4ED0                           jmp     (a0)                    return


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

000000AE 205F           gp_gpiowordin   movea.l (sp)+,a0                pop the return address
000000B0 225F                           movea.l (sp)+,a1                gpio card base address
000000B2 6100 FF60                      bsr     waitready               wait until ready
000000B6 3EA9 0004                      move    4(a1),(sp)              input the datum
000000BA 1280                           move.b  d0,(a1)                 set pctl (same manner as 98032 autohandshake)
000000BC 4ED0                           jmp     (a0)                    return
```



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

000000BE 4E56 0000 gp_gpiodmaout   link    a6,#0                       create our own stack frame

000000C2 226E 0018                 movea.l gpio(a6),a1                 gpio card base address
000000C6 6100 FF4C                 bsr     waitready                   wait for previous handshake to complete

000000CA 4E4B                      trap    #11                         move into supervisor mode       (scs)
                        * scs      move    sr,-(sp)                    prepare to disable interrupts
000000CC 007C 2700                 ori     #$2700,sr                   disable interrupts ******************

000000D0 336E 0016                 move    cmnd(a6),4(a1)              disc command
         0004
000000D6 1280                      move.b  d0,(a1)                     set pctl
000000D8 136E 0014                 move.b  enab(a6),3(a1)              enable the gpio card for dma
         0003

000000DE 246E 0010                 movea.l chan(a6),a2                 dma channel base address
000000E2 24AE 000C                 move.l  buf(a6),(a2)                set the dma address
000000E6 202E 0008                 move.l  len(a6),d0                  transfer length
000000EA 5380                      subq.l  #1,d0                       length-1
000000EC 3540 0004                 move    d0,4(a2)                    set the dma count
000000F0 357C 0006                 move    #$0006,6(a2)                arm the dma channel
         0006

000000F6 46DF                      move    (sp)+,sr                    re-enable interrupts ******************

000000F8 47E9 0007                 lea     7(a1),a3                    gpio register 7 address
000000FC 7003                      moveq   #3,d0                       psts bit (gpio register 7)
000000FE 49EA 0007                 lea     7(a2),a4                    dma status lower byte address
00000102 7200                      moveq   #0,d1                       armed bit (dma status register)

00000104 0113           do_loop    btst    d0,(a3)                     psts?
00000106 6600 FEFA                 bne     ioresc_catchall             ioresc(zcatchall) if so
0000010A 0314                      btst    d1,(a4)                     dma channel still armed?
0000010C 66F6                      bne     do_loop                     keep looping if so

0000010E 0C6A FFFF                 cmpi    #-1,4(a2)                   dma transfer complete normally?
         0004
00000114 6600 FEF6                 bne     bus_error                   branch if not (bus error)

00000118 6100 FEFA                 bsr     waitready                   wait for the final handshake to complete

0000011C 4E5E                      unlk    a6                          remove our stack frame
0000011E 205F                      movea.l (sp)+,a0                    pop the return address
00000120 DEFC 0014                 adda    #20,sp                      pop off the parameters
00000124 4ED0                      jmp     (a0)                        return
```

```
PAGE   7 [3.0] 12/26/84 21:38:26  ASSEMBLY OF GPIODVR.TEXT                                  *** File name: GPIODVR ***

  261                       ********************************************************************************
  262                       *                                  gpiodmain                                   *
  263                       ********************************************************************************
  264
  265    00000126 4E56 0000 gp_gpiodmain    link    a6,#0                   create our own stack frame
  266
  267    0000012A 226E 0018                 movea.l gpio(a6),a1             gpio card base address
  268    0000012E 6100 FEE4                 bsr     waitready               wait for previous handshake to complete
  269
  270    00000132 4E4B                      trap    #11                     move into supervisor mode       (scs)
  271                       * scs            move    sr,-(sp)                prepare to disable interrupts
  272    00000134 007C 2700                 ori     #$2700,sr               disable interrupts ******************
  273
  274    00000138 336E 0016                 move    cmnd(a6),4(a1)          disc command
                  0004
  275    0000013E 1280                      move.b  d0,(a1)                 set pctl
  276
  277    00000140 7000                      moveq   #0,d0                   ready bit (register 0)
  278    00000142 45E9 0007                 lea     7(a1),a2                register 7 address
  279    00000146 7203                      moveq   #3,d1                   psts bit (register 7)
  280
  281    00000148 0312      d_loop          btst    d1,(a2)                 peripheral status?
  282    0000014A 660A                      bne.s   d_enab                  fall out of the critical section if so
  283    0000014C 0111                      btst    d0,(a1)                 ready?
  284    0000014E 67F8                      beq     d_loop                  branch if not
  285
  286    00000150 4269 0004                 clr     4(a1)                   clear the output buffer
  287    00000154 1280                      move.b  d0,(a1)                 set pctl, requesting the first word in
  288
  289    00000156 136E 0014 d_enab          move.b  enab(a6),3(a1)          enable the gpio card for dma
                  0003
  290
  291    0000015C 246E 0010                 movea.l chan(a6),a2             dma channel base address
  292    00000160 24AE 000C                 move.l  buf(a6),(a2)            set the dma address
  293    00000164 357C FFFF                 move    #-1,4(a2)               set count to -1 for the case of one transfer
                  0004
  294    0000016A 202E 0008                 move.l  len(a6),d0              transfer length
  295    0000016E 5580                      subq.l  #2,d0                   length-2
  296    00000170 6D0A                      blt.s   di_reni                 branch if one transfer only
  297    00000172 3540 0004                 move    d0,4(a2)                set the dma count
  298    00000176 357C 0002                 move    #$0002,6(a2)            arm the dma channel
                  0006
  299
  300    0000017C 46DF      di_reni         move    (sp)+,sr                re-enable interrupts ******************
  301
  302    0000017E 47E9 0007                 lea     7(a1),a3                gpio register 7 address
  303    00000182 7003                      moveq   #3,d0                   psts bit (gpio register 7)
  304    00000184 49EA 0007                 lea     7(a2),a4                dma status lower byte address
  305    00000188 7200                      moveq   #0,d1                   armed bit (dma status register)
  306
  307    0000018A 0113      di_loop         btst    d0,(a3)                 psts?
  308    0000018C 6600 FE74                 bne     ioresc_catchall         ioresc(zcatchall) if so
  309    00000190 0314                      btst    d1,(a4)                 dma channel still armed?
  310    00000192 66F6                      bne     di_loop                 keep looping if so
  311
```

```
PAGE   8 [3.0] 12/26/84 21:38:26  ASSEMBLY OF GPIODVR.TEXT                                  *** File name: GPIODVR ***

  313
  314    00000194 0C6A FFFF                 cmpi    #-1,4(a2)               dma transfer complete normally?
                  0004
  315    0000019A 6600 FE70                 bne     bus_error               branch if not (bus error)
  316
  317    0000019E 6100 FE74                 bsr     waitready               wait for last handshake to complete
  318
  319    000001A2 206E 000C                 movea.l buf(a6),a0              buffer address
  320    000001A6 202E 0008                 move.l  len(a6),d0              transfer length in words
  321    000001AA D080                      add.l   d0,d0                   transfer length in bytes
  322    000001AC 31A9 0004                 move    4(a1),-2(a0,d0.l)       transfer last word
                  08FE
  323
  324    000001B2 4E5E                      unlk    a6                      remove our stack frame
  325    000001B4 205F                      movea.l (sp)+,a0                pop the return address
  326    000001B6 DEFC 0014                 adda    #20,sp                  pop off the parameters
  327    000001BA 4ED0                      jmp     (a0)                    return
  328
  329
  330                                       end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

Description

HPIB provides low-level HP-IB drivers.

Usage

Used by the built-in HP-IB and the 98624A card.

```
PAGE   1 [3.0] 12/26/84 21:29:28  ASSEMBLY OF HPIB.TEXT                                  *** File name: HPIB ***

PASS 1 COMPLETE. ERRORS: 0
    2                       ********************************************************************************
    3                       *
    4                       *       COPYRIGHT (C) 1984 BY HEWLETT-PACKARD COMPANY
    5                       *
    6                       ********************************************************************************
    7                       *
    8                       *
    9                       *       IOLIB     EXTH
   10                       *
   11                       *
   12                       ********************************************************************************
   13                       *
   14                       *
   15                       *
   16                       *       Library - IOLIB
   17                       *       Author  -
   18                       *       Phone   -
   19                       *
   20                       *       Purpose - This set of assembly language code is intended to be used as
   21                       *                 a PASCAL module for I/O drivers for use by the external I/O
   22                       *                 procedures library.
   23                       *
   24                       *                 Most of this code is taken from HPL code.
   25                       *
   26                       *       Date    - 08/18/81
   27                       *       Update  - 08/01/83
   28                       *       Release - 10/06/82
   29                       *
   30                       *
   31                       *       Source  - IOLIB:HPIB.TEXT
   32                       *       Object  - IOLIB:HPIB.CODE
   33                       *
   34                       *
   35                       ********************************************************************************
   36                       *
   37                       *
   38                       *       RELEASED
   39                       *       VERSION         3.0
   40                       *
   41                       *
   42                       ********************************************************************************
```

```
PAGE   2 [3.0] 12/26/84 21:29:28 IOLIB EXTH - HPIB DRIVERS                            *** File name: HPIB ***

   44                       ********************************************************************************
   45                       *                                                                              *
   46                       *                                                                              *
   47                       *       BUG FIX HISTORY         - after release 1.0                            *
   48                       *                                                                              *
   49                       *                                                                              *
   50                       *       BUG #   BY  / ON        LOC             DESCRIPTION                    *
   51                       *       -----   ----------      --------------  ----------------------         *
   52                       *                                                                              *
   53                       *       SPR695  ----------      HPL_WTC         the HPIB cards will not        *
   54                       *               04/21/1982                      respond properly to a PPC/     *
   55                       *                                               PPE setup of ppoll info        *
   56                       *                                                                              *
   57                       *       SPR740  ----------      H_TID           DMA input transfers will       *
   58                       *               05/28/1982                      not terminate properly if      *
   59                       *                                               there is an EOI termination    *
   60                       *                                               and EOI is true on the         *
   61                       *                                               first byte and the byte        *
   62                       *                                               comes in immediately.  This    *
   63                       *                                               is a big problem for disk      *
   64                       *                                               transfers.                     *
   65                       *                                                                              *
   66                       *       SPRxxx  ----------      H_BYTTST        DMA input transfers will       *
   67                       *               06/14/1982                      not terminate properly if      *
   68                       *               07/21/1982                      the device is very fast and    *
   69                       *                                               EOI is true on the last        *
   70                       *                                               byte.  Due to DMA on lvl 3     *
   71                       *                                               and an external HPIB card      *
   72                       *                                               on lvl 4,5, or 6.              *
   73                       *                                                                              *
   74                       *       475     ----------      all over        Change BSRs into JSRs to       *
   75                       *               09/17/1982                      allow re-placement of the      *
   76                       *                                               modules.  Also in GPIO and     *
   77                       *                                               Data Comm.                     *
   78                       *                                                                              *
   79                       *       564     ----------      H_WTC_PPC       IOCONTROL(sc,2,x) does not     *
   80                       *               10/22/1982                      work - set up PPOLL byte.      *
   81                       *                                               Always responds with a         *
   82                       *                                               PPOLL response of 4.           *
   83                       *                                                                              *
   84                       ********************************************************************************
   85                       *                                                                              *
   86                       *                                                                              *
   87                       *                                                                              *
   88                       *       BUG FIX HISTORY         - after release 2.0                            *
   89                       *                                                                              *
   90                       *                                                                              *
   91                       *       BUG #   BY  / ON        LOC             DESCRIPTION                    *
   92                       *       -----   ----------      --------------  ----------------------         *
   93                       *                                                                              *
   94                       *       qqqq    ----------      H_TID           Non active controller DMA      *
   95                       *               12/16/1982                      transfers do not work.         *
   96                       *                                               They mess up the count.        *
   97                       *                                                                              *
   98                       *       rrrr    ----------      H_EIR           Re-enabling interrupts when    *
   99                       *               12/17/1982                      srq is already asserted do     *
  100                       *                                               not work.                      *
```

```
101                     *
102                     *       wuwu    -----------     H_DMATERM       Re-enabling interrupts when *
103                     *               01/19/1983                      ON EOT routine is called    *
104                     *                                               due to flukey term emulator *
105                     *                                               problem.                    *
106                     *                                                                           *
107                     *       hphp    -----------     H_INIT_S        Allowing DMA non-ctlr tfrs  *
108                     *               01/28/1983      H_EIR           to be started from inside   *
109                     *                               H_DMATERM       user ISR.                   *
110                     *                               H_ISR                                       *
111                     *                               H_TFR                                       *
112                     *                               H_ENABLE H_DISABLE                          *
113                     *                                                                           *
114                     *       ????    -------         H_T_FHS         Even though FHS with the    *
115                     *               02/02/1983                      internal HPIB to Coyote     *
116                     *                                               makes the interleave, FHS   *
117                     *                                               with the external HPIB to   *
118                     *                                               Coyote does not!!!          *
119                     *                                                                           *
120                     *       tttt    ---------       H_WAIT_BO       Timing changes for 680xx    *
121                     *               08/01/1983      H_WAIT_BI       processors on UMM CPU boards*
122                     *               05/02/1984      H_P_POLL                                    *
123                     *                               H_IFC                                       *
124                     *                               H_ISR0                                      *
125                     *                                                                           *
126                     *****************************************************************************
```



```
128                     *****************************************************************************
129                     *
130                     *
131                     *       The following lines are used to tell the LINKER/LOADER what this module
132                     *       looks like in PASCAL terms.
133                     *
134                     *       Note that it is possible to create assembly modules that are functions.
135                     *       These routines are called through an indirect pointer using the CALL
136                     *       facility which does NOT permit functions.
137                     *
138                     *       This module is called 'EXTH' ( upper or lower case - doesn't matter )
139                     *       independent of the file name ( by use of the MNAME pseudo-op ).
140                     *
141                     *       All the externally used procedures are called 'EXTH_@@@@@@@' in
142                     *       this module.  If you are using assembly to access them use the
143                     *       'EXTH_@@@@@@@' name.  If you are using Pascal use the '@@@@@@@'
144                     *       name.
145                     *
146                     *****************************************************************************
147                             MNAME EXTH
148                             SRC MODULE EXTH;
149                             SRC IMPORT iodeclarations;
150                             SRC EXPORT
151                             SRC             PROCEDURE eh_init   ( temp : ANYPTR );
152                             SRC             PROCEDURE eh_isr    ( temp : ANYPTR );
153                             SRC             PROCEDURE eh_rdb    ( temp : ANYPTR ;  VAR x : CHAR);
154                             SRC             PROCEDURE eh_wtb    ( temp : ANYPTR ;  val   : CHAR);
155                             SRC             PROCEDURE eh_rdw    ( temp : ANYPTR ;  VAR x : io_word);
156                             SRC             PROCEDURE eh_wtw    ( temp : ANYPTR ;  val   : io_word);
157                             SRC             PROCEDURE eh_rds    ( temp : ANYPTR ;  reg   : io_word;
158                             SRC                                                    VAR x : io_word);
159                             SRC             PROCEDURE eh_wtc    ( temp : ANYPTR ;  reg   : io_word;
160                             SRC                                                    val   : io_word );
161                             SRC             PROCEDURE eh_tfr    ( temp : ANYPTR ;  bcb   : ANYPTR );
162                             SRC             PROCEDURE eh_send   ( temp : ANYPTR ;  val   : CHAR );
163                             SRC             PROCEDURE eh_end    ( temp : ANYPTR ;  VAR x : BOOLEAN );
164                             SRC             PROCEDURE eh_ppoll  ( temp : ANYPTR ;  VAR x : CHAR );
165                             SRC             PROCEDURE eh_clr    ( temp : ANYPTR ;  line  : io_bit );
166                             SRC             PROCEDURE eh_set    ( temp : ANYPTR ;  line  : io_bit );
167                             SRC             PROCEDURE eh_test   ( temp : ANYPTR ;  line  : io_bit ;
168                             SRC                                                    VAR x : BOOLEAN );
169                             SRC END; { of EXTH }
```

```
171                     ********************************************************************************
172                     *
173                     *       SYMBOLS FOR EXPORT AS PROCEDURE NAMES
174                     *
175                     ********************************************************************************
176                             DEF EXTH_EXTH
177
178                             DEF EXTH_EH_INIT
179                             DEF EXTH_EH_ISR,EXTH_EH_TDMA
180                             DEF EXTH_EH_RDB,EXTH_EH_WTB
181                             DEF EXTH_EH_RDW,EXTH_EH_WTW
182                             DEF EXTH_EH_RDS,EXTH_EH_WTC
183                             DEF EXTH_EH_TFR
184
185                             DEF EXTH_EH_SEND,EXTH_EH_PPOLL,EXTH_EH_SET
186                             DEF EXTH_EH_CLR,EXTH_EH_TEST,EXTH_EH_END
187
187
187
187
188                     ********************************************************************************
189                     *
190                     *       SYMBOLS FOR IMPORT - COMMON ASSEMBLY LANGUAGE ROUTINES
191                     *
192                     *       THE ROUTINES ARE IN THE MODULE COMMON_ASSEMBLY
193                     *       THE TIMER ROUTINES ARE IN THE FILE "POWERUP"
194                     *
195                     ********************************************************************************
196                             REFA DROPDMA            give up dma resource
197                             REFA GETDMA             actually get dma
198                             REFA TESTDMA            check to see if dma is available
199                             REFA LOGINT             branch to user isr
200                             REFA LOGEOT             branch to user eot
201                             REFA STBSY              set buffer busy
202                             REFA STCLR              set buffer not busy
203                             REFA DMA_STBSY          set buffer dma busy
204                             REFA ITXFR              is there a tfr active ?
205                             REFA ABORT_IO           kill any tfr active
206                             REFA WAIT_TFR           timed wait for tfr active
207                             REFA CHECK_TFR          timed wait for tfr - direction
208                             REFA DELAY_TIMER        timed delay
209                             REFA CHECK_TIMER        timed wait for timeout checking
210
211                     *       change references to allow long jumps when the I/O      475 JPC 9/17/82
212                     *       modules get moved around                                475 JPC 9/17/82
213                             LMODE   DROPDMA,GETDMA,TESTDMA,LOGINT,LOGEOT,STBSY
214                             LMODE   STCLR,DMA_STBSY,ITXFR,ABORT_IO,WAIT_TFR,CHECK_TFR
215                             LMODE   DELAY_TIMER,CHECK_TIMER
216
```



```
219                             INCLUDE IOLIB:COMDCL
```

```
                        ********************************************************************************
                        *
                        *       modified:  02/22/82  JPC    added parm to user EOT & ISR proc's
                        *                  08/01/83  JS     added timer_present and sysflag2 equ'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

0000 0001 TIMER_PRESENT  EQU  1       JS  8/1/83  SYSFLAG2 BIT -- 0=>TIMER PRESENT
FFFF FEDA SYSFLAG2       EQU  $FFFFFEDA         JS  8/1/83
```

```
                        *
0000 0034 H_INT0COPY  EQU AVAIL_OFF+0     COPY OF INT0STAT REGISTER
0000 0035 H_INT1COPY  EQU AVAIL_OFF+1     COPY OF INT1STAT REGISTER
0000 0036 H_INTMSKSAV EQU AVAIL_OFF+2     COPY OF INT0MASK & INT1MASK
0000 0038 H_STAT3     EQU AVAIL_OFF+4     STATUS BYTE 3 MASK:
                        *                     BIT 0:  EOR LATCH
                        *                     BITS 1-7: 0
0000 0039 H_FLAGS     EQU AVAIL_OFF+5     DRIVER FLAGS AND STATUS BYTE 0 MASK:
                        *                     BIT 0:  PASS CONTROL FLAG
                        *                     BIT 1:  USER ISR TO BE CALLED ( IN ISR )
                        *                     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 H_PPOLLMSK  EQU AVAIL_OFF+6     VALUE TO PUT IN H_PPOLL WHEN ist = 1
                        *     EQU AVAIL_OFF+7     VALUE TO PUT IN H_PPOLL WHEN ist = 0
```



```
                        ********************************************************************************
                        *
                        *       PASCAL DRIVER ENTRY POINTS FOR HP-IB CARDS
                        *
                        ********************************************************************************

                        *
                        *       Module initialization
                        *
          0000 0000 EXTH_EXTH EQU *
00000000 4E75               RTS                 Do nothing
                        *
                        *       Driver initialization
                        *
          0000 0002 EXTH_EH_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 0148          BRA     H_INIT
                        *
                        *       Interrupt service routine
                        *
          0000 0010 EXTH_EH_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 0754          BRA     H_ISR
                        *
                        *       HPIB DMA transfer termination routine
                        *
          0000 001E EXTH_EH_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 06E8          BRA     H_DMATERM
                        *
                        *       Read a byte
                        *
          0000 002C EXTH_EH_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 01AE          BSR     H_RDB       call read byte
0000003C 1680               MOVE.B  D0,(A3)     save character
0000003E 4E75               RTS
                        *
                        *       Write a byte
                        *
          0000 0040 EXTH_EH_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 01C8          BRA     H_WTB         call write byte
                        *
                        *       Read a word
                        *
                0000 0050 EXTH_EH_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 018A          BSR     H_RDB         call read byte
00000060 1A00               MOVE.B  D0,D5         save byte
00000062 6100 0184          BSR     H_RDB         read second byte
00000066 E14D               LSL.W   #8,D5         shift first by
00000068 1A00               MOVE.B  D0,D5         bring in low bits
0000006A 3685               MOVE.W  D5,(A3)       save word
0000006C 4E75               RTS
                        *
                        *       Write a word
                        *
                0000 006E EXTH_EH_WTW     EQU *
0000006E 205F               MOVEA.L (SP)+,A0      get return address
00000070 301F               MOVE.W  (SP)+,D0      get word value
00000072 245F               MOVEA.L (SP)+,A2      get temp address
00000074 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
00000078 4850               PEA     (A0)          push return address back on stack
0000007A 1A00               MOVE.B  D0,D5         save second byte
0000007C E048               LSR     #8,D0
0000007E 6100 0196          BSR     H_WTB         write the byte
00000082 1005               MOVE.B  D5,D0         get the second byte
00000084 6000 0190          BRA     H_WTB         write the byte
                        *
                        *       Read status
                        *
                0000 0088 EXTH_EH_RDS     EQU *
00000088 205F               MOVEA.L (SP)+,A0      get return address
0000008A 265F               MOVEA.L (SP)+,A3      get VAR address
0000008C 321F               MOVE.W  (SP)+,D1      get register number
0000008E 245F               MOVEA.L (SP)+,A2      get temp address
00000090 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
00000094 4850               PEA     (A0)          push return address back on stack
00000096 6100 030E          BSR     H_RDS         read status
0000009A 3680               MOVE.W  D0,(A3)       save status info
0000009C 4E75               RTS
                        *
                        *       Write control
                        *
                0000 009E EXTH_EH_WTC     EQU *
0000009E 205F               MOVEA.L (SP)+,A0      get return address
000000A0 301F               MOVE.W  (SP)+,D0      get value
000000A2 321F               MOVE.W  (SP)+,D1      get register number
000000A4 245F               MOVEA.L (SP)+,A2      get temp address
000000A6 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
000000AA 4850               PEA     (A0)          push return address back or stack
000000AC 6000 037E          BRA     H_WTC         write control
```



```
                        *
                        *       Transfer
                        *
                0000 00B0 EXTH_EH_TFR     EQU *
000000B0 205F               MOVEA.L (SP)+,A0      get return address
000000B2 265F               MOVEA.L (SP)+,A3      get buffer control block address
000000B4 245F               MOVEA.L (SP)+,A2      get temp address
000000B6 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
000000BA 4850               PEA     (A0)          push return address back on stack
000000BC 6000 09FE          BRA     H_TFR         transfer
                        *
                        *       Send an 'ATN' true command
                        *
                0000 00C0 EXTH_EH_SEND    EQU *
000000C0 205F               MOVEA.L (SP)+,A0      get return address
000000C2 101F               MOVE.B  (SP)+,D0      get value          ( this actually bumps SP by 2 )
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)          push return address back on stack
000000CC 6000 05C8          BRA     H_R60UT       send command byte
                        *
                        *       Perform a Parallel Poll
                        *
                0000 00D0 EXTH_EH_PPOLL   EQU *
000000D0 205F               MOVEA.L (SP)+,A0      get return address
000000D2 265F               MOVEA.L (SP)+,A3      get VAR address
000000D4 245F               MOVEA.L (SP)+,A2      get temp address
000000D6 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
000000DA 4850               PEA     (A0)          push return address back on stack
000000DC 6100 0490          BSR     H_P_POLL      do a parallel poll
000000E0 1680               MOVE.B  D0,(A3)       save value
000000E2 4E75               RTS
                        *
                        *       Set an HPIB line
                        *
                0000 00E4 EXTH_EH_SET     EQU *
000000E4 205F               MOVEA.L (SP)+,A0      get return address
000000E6 321F               MOVE.W  (SP)+,D1      get line           ( this actually bumps SP by 2 )
000000E8 245F               MOVEA.L (SP)+,A2      get temp address
000000EA 226A 0020          MOVEA.L C_ADR(A2),A1  get card address
000000EE 4850               PEA     (A0)          push return address back on stack
000000F0 6000 04B0          BRA     H_SET         call set line
                        *
                        *       Clear an HPIB line
                        *
                0000 00F4 EXTH_EH_CLR     EQU *
000000F4 205F               MOVEA.L (SP)+,A0      get return address
000000F6 321F               MOVE.W  (SP)+,D1      get line           ( this actually bumps SP by 2 )
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 6000 0540          BRA     H_CLR         clear the line
                        *
                        *       Test an HPIB line
                        *
                0000 0104 EXTH_EH_TEST    EQU *
00000104 205F               MOVEA.L (SP)+,A0      get return address
```

```
00000106 265F                MOVEA.L (SP)+,A3      get VAR address
00000108 321F                MOVE.W  (SP)+,D1      get line          ( this actually bumps SP by 2 )
0000010A 245F                MOVEA.L (SP)+,A2      get temp address
0000010C 226A 0020           MOVEA.L C_ADR(A2),A1  get card address
00000110 4850                PEA     (A0)          push return address back on stack
00000112 6100 0576           BSR     H_TEST        read status
00000116 1680                MOVE.B  D0,(A3)       save character
00000118 4E75                RTS
                     *
                     *       Test for EOI/END condition
                     *
            0000 011A EXTH_EH_END    EQU *
0000011A 205F                MOVEA.L (SP)+,A0      get return address
0000011C 265F                MOVEA.L (SP)+,A3      get VAR address
0000011E 245F                MOVEA.L (SP)+,A2      get temp address
00000120 226A 0020           MOVEA.L C_ADR(A2),A1  get card address
00000124 4850                PEA     (A0)          push return address back on stack
00000126 102A 0038           MOVE.B  H_STAT3(A2),D0 get EOR bit
0000012A 0200 0001           ANDI.B  #1,D0         mask it off
0000012E 1680                MOVE.B  D0,(A3)       save condition
00000130 4E75                RTS
```



```
                     ********************************************************************************
                     *
                     *       ADDRESS CONSTANTS
                     *
                     ********************************************************************************
            0047 8000 H_INT_CA    EQU $478000         address of internal HP-IB card


                     ********************************************************************************
                     *
                     *       HP-IB CARD ADDRESS EQUATES  ( OFFSETS FROM A1 )
                     *
                     *           for the TI 9914
                     *
                     *           HPL DECLARATIONS
                     *
                     ********************************************************************************
            0000 0005 H_EXTSTAT   EQU $05   READ      EXTERNAL STATUS REGISTER
            0000 0011 H_INT0STAT  EQU $11   READ      INTERRUPT STATUS REGISTER 0
            0000 0011 H_INT0MASK  EQU $11   WRITE     INTERRUPT MASK REGISTER 0
            0000 0013 H_INT1STAT  EQU $13   READ      INTERRUPT STATUS REGISTER 1
            0000 0013 H_INT1MASK  EQU $13   WRITE     INTERRUPT MASK REGISTER 1
            0000 0015 H_ADRSSTAT  EQU $15   READ      ADDRESS STATUS REGISTER
            0000 0017 H_BUSSTAT   EQU $17   READ      BUS STATUS REGISTER
            0000 0017 H_AUXCMD    EQU $17   WRITE     AUXILLARY COMMAND REGISTER
            0000 0019 H_ADDRESS   EQU $19   WRITE     ADDRESS REGISTER
            0000 001B H_SPOLL     EQU $1B   WRITE     SERIAL POLL RESPONSE REGISTER
            0000 001D H_CMDPASS   EQU $1D   READ      COMMAND PASS THROUGH REGISTER
            0000 001D H_PPOLL     EQU $1D   WRITE     PARALLEL RESPONSE REGISTER
            0000 001F H_DATAIN    EQU $1F   READ      DATA IN REGISTER
            0000 001F H_DATAOUT   EQU $1F   WRITE     DATA OUT REGISTER
```

```
                        ********************************************************************************
                        *
                        *       HP-IB AUXILLARY COMMAND EQUATES
                        *
                        *           for the TI 9914
                        *
                        *           HPL DECLARATIONS
                        *
                        ********************************************************************************
0000 0000 H_SWRST0      EQU $00   FALSE     SOFTWARE RESET
0000 0080 H_SWRST1      EQU $80   TRUE        "       "
0000 0001 H_DACR0       EQU $01   FALSE     RELEASE DAC HOLDOFF
0000 0081 H_DACR1       EQU $81   TRUE        "       "       "
0000 0002 H_RHDF        EQU $02   PULSE     RELEASE RFD HOLDOFF
0000 0003 H_HDFA0       EQU $03   FALSE     HOLDOFF ON ALL DATA
0000 0083 H_HDFA1       EQU $83   TRUE        "       "   "   "
0000 0004 H_HDFE0       EQU $04   FALSE     HOLDOFF ON END
0000 0084 H_HDFE1       EQU $84   TRUE        "       "   "
0000 0005 H_NBAF        EQU $05   PULSE     SET NEW BYTE AVAILABLE
0000 0006 H_FGET0       EQU $06   FALSE     FORCE GROUP EXECUTE TRIGGER
0000 0086 H_FGET1       EQU $86   TRUE        "       "       "       "
0000 0007 H_RTL0        EQU $07   FALSE     RETURN TO LOCAL
0000 0087 H_RTL1        EQU $87   TRUE        "       "       "
0000 0008 H_FEOI        EQU $08   PULSE     FORCE EOI
0000 0009 H_LON0        EQU $09   FALSE     LISTEN ONLY
0000 0089 H_LON1        EQU $89   TRUE        "       "
0000 000A H_TON0        EQU $0A   FALSE     TALK ONLY
0000 008A H_TON1        EQU $8A   TRUE        "       "
0000 000B H_GTS         EQU $0B   PULSE     GO TO STANBY
0000 000C H_TCA         EQU $0C   PULSE     TAKE CONTROL ASYNCHRONOUSLY
0000 000D H_TCS         EQU $0D   PULSE     TAKE CONTROL SYNCHRONOUSLY
0000 000E H_RPP0        EQU $0E   FALSE     REQUEST PARALLEL POLL
0000 008E H_RPP1        EQU $8E   TRUE        "       "       "
0000 000F H_SIC0        EQU $0F   FALSE     SEND IFC
0000 008F H_SIC1        EQU $8F   TRUE        "    "
0000 0010 H_SRE0        EQU $10   FALSE     SEND REN
0000 0090 H_SRE1        EQU $90   TRUE        "    "
0000 0011 H_RQC         EQU $11   PULSE     REQUEST CONTROL
0000 0012 H_RLC         EQU $12   PULSE     RELEASE CONTROL
0000 0013 H_DAI0        EQU $13   FALSE     DISABLE ALL INTERRUPTS
0000 0093 H_DAI1        EQU $93   TRUE        "       "       "
0000 0014 H_PTS         EQU $14   PULSE     PASS THROUGH NEXT SECONDARY
0000 0015 H_STDL0       EQU $15   FALSE     SET T1 DELAY (1200ns)
0000 0095 H_STDL1       EQU $95   TRUE        "   "   "
0000 0016 H_SHDW0       EQU $16   FALSE     SHADOW HANDSHAKE
0000 0096 H_SHDW1       EQU $96   TRUE        "       "
0000 0017 H_VSTDL0      EQU $17   FALSE     SPECIAL SET T1 DELAY FOR 9914A (600ns)
0000 0097 H_VSTDL1      EQU $97   TRUE        "   "   "   "
```



```
                        ********************************************************************************
                        *
                        *       HP-IB command equates
                        *
                        *           PASCAL DECLARATIONS
                        *
                        ********************************************************************************
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
```

```
                        *                                                       wuwu TM 1/19/83
                        * SET THE PROCESSOR INTERRUPT LEVEL TO THE INTERFACE    wuwu TM 1/19/83
                        *         CARD'S INTERRUPT LEVEL                        wuwu TM 1/19/83
                        *                                                       wuwu TM 1/19/83
                        *       A1 MUST HAVE THE CARD ADDRESS                   wuwu TM 1/19/83
                        *                                                       wuwu TM 1/19/83
00000132 7000           SET_INT_LEVEL   MOVEQ   #0,D0                         wuwu TM 1/19/83
00000134 B3FC 0047 8000                 CMPA.L  #$478000,A1     THIS THE INTERNAL HPIB?
0000013A 6708                           BEQ.S   INTLEV_1        BRANCH IF SO
0000013C 7030                           MOVEQ   #$30,D0         CARD'S INTERRUPT LEVEL MASK
0000013E C029 0003                      AND.B   3(A1),D0        INTERRUPT LEVEL IN UPPER NIBBLE
00000142 E848                           LSR     #4,D0           SHIFT TO LOWER NIBBLE
00000144 5640           INTLEV_1        ADDQ    #3,D0           CONVERT TO PROCESSOR'S INTERRUPT LEVEL
00000146 E148                           LSL     #8,D0           SHIFT TO UPPER BYTE
00000148 3F00                           MOVE    D0,-(SP)        SAVE FOR A MOMENT
0000014A 40C0                           MOVE    SR,D0           CURRENT STATUS REGISTER
0000014C 0240 F8FF                      ANDI    #$F8FF,D0       STRIP CURRENT INT LEVEL BITS
00000150 805F                           OR      (SP)+,D0        SUBSTITUTE NEW INT LEVEL BITS
00000152 46C0                           MOVE    D0,SR           SET NEW INTERRUPT LEVEL
00000154 4E75                           RTS                                     wuwu TM 1/19/83
```



```
                        ********************************************************************************
                        *
                        *       H_INIT
                        *
                        *         INITIALIZE AN HP-IB CARD
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
            0000 0156 H_INIT    EQU     *
                        * tm      MOVE.B  MA(A2),D0               ASSUME THIS IS NOT POWER UP AND
00000156 303C 0015              MOVE.W  #21,D0                  ASSUME THIS IS THE INTERNAL CARD
0000015A 0829 0007              BTST    #7,H_EXTSTAT(A1)        SO CHOOSE ADDRESS 21 IF THIS
0005
00000160 6604                   BNE.S   H_INIT_C                IS SYSTEM CONTROLLER ELSE
00000162 303C 0014              MOVE.W  #20,D0                  CHOOSE ADDRESS 20.
00000166 B3FC 0047 H_INIT_C     CMPA.L  #H_INT_CA,A1            IS THIS THE INTERNAL CARD?
8000
0000016C 6710                   BEQ.S   H_INIT0                 IF SO, SKIP
0000016E 1029 0005              MOVE.B  H_EXTSTAT(A1),D0        ELSE GET ADDRESS FROM CARD
00000172 C07C 001F              AND     #$1F,D0
00000176 B07C 001F              CMP     #31,D0                  IF CARD SAYS IT IS AT ADDRESS
0000017A 6602                   BNE.S   H_INIT0                 31, THEN USE ZERO INSTEAD!
0000017C 7000                    MOVEQ #0,D0
0000017E 137C 0010 H_INIT0      MOVE.B  #H_SRE0,H_AUXCMD(A1)    set REN false
0017
00000184 6106                   BSR.S   H_INIT_S                START SOFTWARE RESET
00000186 6600 044A              BNE     H_IFC                   IF SYSTEM CONTROLLER, BRANCH
0000018A 4E75                   RTS
                        ********************************************************************************
                        *
                        *       H_INIT_S
                        *
                        *         SUBROUTINE USED FOR BOTH  INITIALIZATION AND wtc:
                        *
                        *         HPL ROUTINE
                        *
                        ********************************************************************************
0000018C 137C 0080 H_INIT_S     MOVE.B  #H_SWRST1,H_AUXCMD(A1) START SOFTWARE RESET
0017
00000192 3540 0032              MOVE.W  D0,MA_W(A2)             SAVE MY ADDRESS
00000196 1340 0019              MOVE.B  D0,H_ADDRESS(A1)        AND TELL CARD MY ADDRESS
0000019A 4EB9 0000              JSR     ABORT_IO                CLEANUP ANY ATTACHED BUFFER
0000
000001A0 41E9 0017              LEA     H_AUXCMD(A1),A0         MAKE A0 POINT TO AUX CMD REG
000001A4 7000                   MOVEQ   #0,D0                   AND PRELOAD D0 WITH A ZERO
000001A6 10BC 0095              MOVE.B  #H_STDL1,(A0)           SET T1 DELAY (1200NS)
000001AA 10BC 0097              MOVE.B  #H_VSTDL1,(A0)          SET T1 DELAY FOR 9914A (600NS)
000001AE 0189 0011              MOVEP   D0,H_INTOMASK(A1)       FOR NOW, CLEAR BOTH INT MASKS
000001B2 3540 0034              MOVE.W  D0,H_INTOCOPY(A2)       CLEAR COPYS OF INT STAT REGS
000001B6 3540 0038              MOVE.W  D0,H_STAT3(A2)          INIT. DRIVER FLAGS
000001BA 10BC 0083              MOVE.B  #H_HDFA1,(A0)           SET HOLD OFF ON ALL DATA
000001BE 10BC 0004              MOVE.B  #H_HDFE0,(A0)           CLEAR HOLD OFF ON END
000001C2 10BC 000E              MOVE.B  #H_RFPO,(A0)            CLEAR PAR. POLL IF ACTIVE.
000001C6 1340 001B              MOVE.B  D0,H_SPOLL(A1)          CLEAR SERIAL POLL RESPONSE
000001CA 1340 001D              MOVE.B  D0,H_PPOLL(A1)          UNCONFIGURE PARALLEL POLL
000001CE 3540 003A              MOVE.W  D0,H_PPOLLMSK(A2)       CLEAR PPOLL MASK
```

```
PAGE  21 [3.0] 12/26/84 21:29:28 IOLIB EXTH - HPIB DRIVERS                                  *** File name: HPIB ***

  793    000001D2 10BC 0000            MOVE.B #H_SWRST0,(A0)          CLEAR SOFTWARE RESET
  794                          * tm     MOVEQ  #0,D0                   SET UP INT MASKS
  795    000001D6 6100 02EC            BSR    H_EIR
  796    000001DA 137C 0080            MOVE.B #$80,3(A1)              ENABLE THE CARD       ( hphp TM 1/19/83 )
                  0003
  797    000001E0 0829 0007            BTST   #7,H_EXTSTAT(A1)       IS THIS A SYSTEM CONTROLLER?
                  0005
  798    000001E6 4E75                 RTS                            (LEAVE CC FOR CALLER)
```

```
PAGE  22 [3.0] 12/26/84 21:29:28 IOLIB EXTH - HPIB DRIVERS                                  *** File name: HPIB ***

  800                          ********************************************************************************
  801                          *
  802                          *       H_RDB
  803                          *
  804                          *         READ A BYTE OF DATA FROM HP-IB
  805                          *
  806                          *         EXIT:   D0.B = BYTE READ
  807                          *
  808                          *         HPL ROUTINE
  809                          *
  810                          ********************************************************************************
  811    00001E8 137C 000B H_RDB       MOVE.B #H_GTS,H_AUXCMD(A1)     CLEAR ATN
                  0017
  812    00001EE 0829 0002             BTST   #2,H_ADRSSTAT(A1)       MAKE SURE ADDRESSED TO LISTEN
                  0015
  813    00001F4 6754                  BEQ.S  H_LSTERR                ELSE GIVE ERROR
  814    00001F6 08AA 0007 H_RDB0      BCLR   #7,H_FLAGS(A2)          TEST (AND CLEAR) HOLDOFF FLAG
                  0039
  815    00001FC 670C                  BEQ.S  H_RDB1                  IF IT WAS CLEAR, SKIP
  816    00001FE 08AA 0000             BCLR   #0,H_STAT3(A2)          CLEAR EOR ( EOI ) FLAG IN TEMPS
                  0038
  817    0000204 137C 0002             MOVE.B #H_RHDF,H_AUXCMD(A1)    RELEASE RFD HOLDOFF TO START HS
                  0017
  818    000020A 6100 00F0 H_RDB1      BSR    H_WAIT_BI               NOW WAIT FOR BYTE IN
  819    000020E 7000                  MOVEQ  #0,D0                   ELSE CLEAR UPPER PART OF D0
  820    0000210 1029 001F             MOVE.B H_DATAOUT(A1),D0        AND PUT DATA IN LOWER BYTE
  821    0000214 4E75                  RTS                            DONE!
  822
  822
  822
  822
  822
  823                          ********************************************************************************
  824                          *
  825                          *       H_WTB
  826                          *
  827                          *         WRITE A BYTE OF DATA TO HP-IB
  828                          *
  829                          *         ENTRY:  D0.B = BYTE TO WRITE
  830                          *
  831                          *         HPL ROUTINE
  832                          *
  833                          ********************************************************************************
  834    0000216 0829 0001 H_WTB       BTST   #1,H_ADRSSTAT(A1)       MAKE SURE ADDRESSED TO TALK
                  0015
  835    000021C 6730                  BEQ.S  H_TLKERR                ELSE ERROR
  836    000021E 137C 000B H_WTB0      MOVE.B #H_GTS,H_AUXCMD(A1)     CLEAR ATN
                  0017
  837    0000224 6100 0044 H_WTB1      BSR    H_WAIT_BO               WAIT FOR BYTE OUT
  838    0000228 1340 001F             MOVE.B D0,H_DATAOUT(A1)        MOVE THE DATA OUT
  839    000022C 4E75                  RTS                            DONE!
  840
  840
  840
```

```
                        ********************************************************************************
                        *
                        *       Error escapes
                        *
                        ********************************************************************************
0000022E 7008           H_SCBSY   MOVEQ  #SC_BUSY,D0          buffer is busy
00000230 6022                     BRA.S  ESC_ERR
00000232 7014           H_SC_ERR  MOVEQ  #BAD_SCT,D0          bad set/clear/test
00000234 601E                     BRA.S  ESC_ERR
00000236 7003           H_NOTACTL MOVEQ  #NO_ACTL,D0          not active controller
00000238 601A                     BRA.S  ESC_ERR
0000023A 7012           H_NOTSCTL MOVEQ  #NO_SCTL,D0          not system controller
0000023C 6016                     BRA.S  ESC_ERR
0000023E 7007           HTERR_B   MOVEQ  #TFR_ERR,D0          bad transfer specification
00000240 6012                     BRA.S  ESC_ERR
00000242 700D           HTERR_D   MOVEQ  #NO_DMA,D0           DMA not installed
00000244 600E                     BRA.S  ESC_ERR
00000246 700E           H_NOWORD  MOVEQ  #NO_WORD,D0          WORD transfers not allowed
00000248 600A                     BRA.S  ESC_ERR
0000024A 7010           H_LSTERR  MOVEQ  #NOT_LSTN,D0         not addressed as listener
0000024C 6006                     BRA.S  ESC_ERR
0000024E 700F           H_TLKERR  MOVEQ  #NOT_TALK,D0         not addressed as talker
00000250 6002                     BRA.S  ESC_ERR
00000252 7011           H_TMO     MOVEQ  #TMO_ERR,D0          timeout
                        *         BRA.S  ESC_ERR


00000254 48C0           ESC_ERR   EXT.L  D0
00000256 2B40 FFBE                MOVE.L D0,IOE_RSLT(A5)        save error in io space
0000025A 102A 002D                MOVE.B IO_SC(A2),D0         \ get sc for error
0000025E 2B40 FFBA                MOVE.L D0,IOE_SC(A5)        /
00000262 3B7C FFE6                MOVE.W #IOE_ERROR,ESC_CODE(A5)  save system esc code
         FFFE
00000268 4E4A                     TRAP   #10                    escape
```



```
                        ********************************************************************************
                        *
                        *       HP-IB WAIT ROUTINES
                        *
                        *
                        *         ENTRY:  H_WAIT_BO      WAIT FOR BO STATUS TO BE TRUE
                        *                 H_WAIT_BI      WAIT FOR BI STATUS TO BE TRUE
                        *
                        *         EXIT:   IF CONDITION IS OR COMES TRUE, RTS.
                        *                 THE ERROR ESCAPE IS GENERATED
                        *                   IF TIMEOUT > 0  AND <TIMEOUT> MS HAS EXPIRED, OR
                        *
                        *         NOTE:   DURING THE FIRST 1-2 MS OF THE WAIT, A QUICK CHECK ALGORITHM IS
                        *                 USED WHICH DOES NOT CHECK THE TIMEOUT - THUS IF
                        *                 THE DATA RATE IS > 1 KB, NO TIMEOUT DETECTION OVERHEAD OCCURS.
                        *
                        *
                        *         HPL ROUTINE ( MODIFIED )
                        *
                        ********************************************************************************
0000026A 243C 0000 H_WAIT_BO MOVE.L #254,D2               D2 = QUICK CHECK LOOP COUNTER
         00FE
                        *
                        *  Quick check counter was 127, changed to 254    tttt  JS 8/1/83
                        *  ALSO CHANGED MOVEQ TO MOVE.L
                        *
00000270 1229 0011 H_WBO_1   MOVE.B H_INTOSTAT(A1),D1      GET THE INTERRUPT STATUS
00000274 B27C 003F           CMP    #$3F,D1              IF IN READING THE STATUS WE MISSED AN
00000278 6306                BLS.S  H_WBO_2              AN INTERRUPT, THEN WE HAVE TO
0000027A 6100 0110            BSR   H_FAKEISR           FAKE AISR CALL...DUMB HARDWARE!
0000027E 7200                 MOVEQ #0,D1
00000280 822A 0034 H_WBO_2   OR.B   H_INTOCOPY(A2),D1      THIS IS IN CASE ISR LEFT STUFF HERE
00000284 0801 0004           BTST   #4,D1                BYTE OUT?
00000288 662E                BNE.S  H_W_DONE             IF SO, GET OUT!
0000028A 51CA FFE4           DBRA   D2,H_WBO_1           ELSE LOOP BACK

0000028E 242A 002E           MOVE.L TIMEOUT(A2),D2       OK, SET UP TO WATCH FOR TIMEOUT,ETC
00000292 671A                BEQ.S  H_WBO_5                if =0 goto inf loop
00000294 0838 0001           BTST   #TIMER_PRESENT,SYSFLAG2  CHECK IF TIMER THERE  tttt JS 8/1/83
         FEDA
0000029A 673C                BEQ.S  H_WBOT                 YES, USE IT             tttt JS 8/1/83
                        * tm    MULU   #60,D2               60 TIMES THROUGH LOOP = 1 MS
0000029C ED8A                LSL.L  #6,D2                ( * 64 IS CLOSE ENOUGH )
0000029E 6100 00D4 H_WBO_3   BSR    H_GETSTAT            GO GET STATUS
000002A2 0801 0004           BTST   #4,D1                BYTE OUT?
000002A6 6614                BNE.S  H_WAIT_D1            YES, GET OUT OF HERE!
000002A8 5382                SUBQ.L #1,D2                LOOP UNTIL GRACE PERIOD DONE
000002AA 66F2                BNE    H_WBO_3
000002AC 6022                BRA.S  H_TMO_ERR            GIVE ERROR

000002AE 6100 00C4 H_WBO_5   BSR    H_GETSTAT            ELSE TRY AGAIN FOR STATUS
000002B2 0801 0004           BTST   #4,D1
000002B6 67F6                BEQ    H_WBO_5              IF NOT SET, KEEP WAITING

000002B8 832A C034 H_W_DONE  OR.B   D1,H_INTOCOPY(A2)    SAVE ANY STATUS BITS WE DIDN'T USE
000002BC 022A C0CF H_WAIT_D1 ANDI.B #$CF,H_INTOCOPY(A2)  CLEAR BO/BI BITS
         0034
```

```
000002C2 0801 0005          BTST   #5,D1              DID WE GET A BYTE IN?
000002C6 6706               BEQ.S  H_WAIT_D2         IF NOT, SKIP
000002C8 08EA 0007          BSET   #7,H_FLAGS(A2)    ELSE SET HOLDOFF FLAG
         0039
000002CE 4E75       H_WAIT_D2 RTS                    DONE!


000002D0 832A 0034 H_TMO_ERR OR.B   D1,H_INTOCOPY(A2)   SAVE ANY STATUS BITS NOT USED.
000002D4 6000 FF7C          BRA    H_TMO
000002D8 1F3C 0001 H_WBOT   MOVE.B #1,-(SP)          SETUP TIMER RECORD    tttt JS 8/1/83
000002DC 2F02               MOVE.L D2,-(SP)                             tttt JS 8/1/83
000002DE 6100 0094 H_WBOT1  BSR    H_GETSTAT         CHECK FOR BYTE OUT    tttt JS 8/1/83
000002E2 0801 0004          BTST   #4,D1             BO SET?               tttt JS 8/1/83
000002E6 6610               BNE.S  H_WBOT2           YES, GET OUT OF HERE  tttt JS 8/1/83
000002E8 4857               PEA    (SP)              ELSE CHECK TIMER      tttt JS 8/1/83
000002EA 4EB9 0000          JSR    CHECK_TIMER                             tttt JS 8/1/83
         0000
000002F0 6AEC               BPL    H_WBOT1           BR IF NOT TIMED OUT   tttt JS 8/1/83
000002F2 5C4F               ADDQ   #6,SP             TIMEOUT, GIVE ONE     tttt JS 5/2/84
000002F4 743C               MOVEQ  #60,D2            MORE CHANCE WITH      tttt JS 5/2/84
000002F6 60A6               BRA    H_WBO_3           SHORT TIMEOUT         tttt JS 5/2/84
000002F8 5C4F       H_WBOT2 ADDQ   #6,SP             CLEAN UP STACK        tttt JS 8/1/83
000002FA 60C0               BRA    H_WAIT_D1         AND RETURN            tttt JS 8/1/83


000002FC 243C 0000 H_WAIT_BI MOVE.L  #254,D2          D2 = QUICK CHECK LOOP COUNTER
         00FE
                    *
                    * Quick timeout count was 127, changed to get 1 MS on 16 MHz processor
                    *    tttt  JS 8/1/83  ALSO CHANGED MOVEQ TO MOVE.L
                    *
00000302 1229 0011 H_WBI_1  MOVE.B H_INTOSTAT(A1),D1  GET THE INTERRUPT STATUS
00000306 B27C 003F          CMP    #$3F,D1           IF IN READING THE STATUS WE MISSED AN
0000030A 6306               BLS.S  H_WBI_2           AN INTERRUPT, THEN WE HAVE TO
0000030C 6100 007E           BSR   H_FAKEISR         FAKE AISR CALL...DUMB HARDWARE!
00000310 7200                MOVEQ #0,D1
00000312 822A 0034 H_WBI_2  OR.B   H_INTOCOPY(A2),D1  THIS IS IN CASE ISR LEFT STUFF HERE
00000316 0801 0005          BTST   #5,D1             BYTE IN?
0000031A 669C               BNE.S  H_W_DONE          IF SO, GET OUT!
0000031C 51CA FFE4          DBRA   D2,H_WBI_1        ELSE LOOP BACK

00000320 242A 002E          MOVE.L TIMEOUT(A2),D2    OK, SET UP TO WATCH FOR TIMEOUT,ETC
00000324 671A               BEQ.S  H_WBI_5             if =0 goto inf loop
00000326 0838 0001          BTST   #TIMER_PRESENT,SYSFLAG2  CHECK FOR TIMER     tttt JS 8/1/83
         FEDA
0000032C 6720               BEQ.S  H_WBIT              IF THERE USE IT      tttt JS 8/1/83
                    * tm    MULU   #60,D2            60 TIMES THROUGH LOOP = 1 MS
0000032E ED8A               LSL.L  #6,D2             ( * 64 IS CLOSE ENOUGH )
00000330 6100 0042 H_WBI_3  BSR    H_GETSTAT         GO GET STATUS
00000334 0801 0005          BTST   #5,D1             BYTE IN?
00000338 6682               BNE.S  H_WAIT_D1         YES, GET OUT OF HERE!
0000033A 5382               SUBQ.L #1,D2             LOOP UNTIL GRACE PERIOD DONE
0000033C 66F2               BNE    H_WBI_3
```



```
0000033E 6090               BRA.S  H_TMO_ERR           IF SO, GIVE ERROR

00000340 6100 0032 H_WBI_5  BSR    H_GETSTAT         ELSE TRY AGAIN FOR STATUS
00000344 0801 0005          BTST   #5,D1
00000348 6600 FF72          BNE    H_WAIT_D1         IF SET, GET OUT!
0000034C 60F2               BRA    H_WBI_5
0000034E 1F3C 0001 H_WBIT   MOVE.B #1,-(SP)          SET UP TIMER RECORD   tttt JS 8/1/83
00000352 2F02               MOVE.L D2,-(SP)                             tttt JS 8/1/83
00000354 6100 001E H_WBIT1  BSR    H_GETSTAT         GET STATUS            tttt JS 8/1/83
00000358 0801 0005          BTST   #5,D1             CHECK FOR BI SET      tttt JS 8/1/83
0000035C 6610               BNE.S  H_WBIT2           IF GOTIT THEN EXIT    tttt JS 8/1/83
0000035E 4857               PEA    (SP)              ELSE CHECK TIMER      tttt JS 8/1/83
00000360 4EB9 0000          JSR    CHECK_TIMER                             tttt JS 8/1/83
         0000
00000366 6AEC               BPL    H_WBIT1           BR IF NOT TIMED OUT   tttt JS 8/1/83
00000368 5C4F               ADDQ   #6,SP             TIMEOUT, GIVE ONE MORE tttt JS 5/2/84
0000036A 743C               MOVEQ  #60,D2            CHANCE WITH SHORT     tttt JS 5/2/84
0000036C 60D2               BRA    H_WBI_3           TIMEOUT COUNT         tttt JS 5/2/84
0000036E 5C4F       H_WBIT2 ADDQ   #6,SP             CLEAN UP TIMER RECORD tttt JS 8/1/83
00000370 6000 FF4A          BRA    H_WAIT_D1         AND GET OUT           tttt JS 8/1/83
```

```
 997                    ********************************************************************************
 998                    *
 999                    *       H_GETSTAT
1000                    *
1001                    *         SUBROUTINE TO GET INTOSTAT AND INSURE WE DON'T MISS AN
1002                    *             INTERRUPT
1003                    *
1004                    *         HPL ROUTINE
1005                    *
1006                    ********************************************************************************
1007  00000374 1229 0011 H_GETSTAT MOVE.B H_INTOSTAT(A1),D1       GET CURRENT INTERRUPT STATUS
1008  00000378 B23C 003F           CMP.B  #$3F,D1                 DID WE MISS AN INTERRUPT?
1009  0000037C 6304                BLS.S  H_G_STAT1               IF NOT, THEN DONE
1010  0000037E 610C                BSR.S  H_FAKEISR               ELSE FAKE AN ISR CALL
1011  00000380 7200                MOVEQ  #0,D1                   AND JUST USE THE COPY
1012  00000382 822A 0034 H_G_STAT1 OR.B   H_INTOCOPY(A2),D1      INCLUDE ANY SAVED BITS
1013  00000386 832A 0034           OR.B   D1,H_INTOCOPY(A2)
1014  0000038A 4E75                RTS
1015
1015
1016                    ********************************************************************************
1017                    *
1018                    *       H_FAKEISR
1019                    *
1020                    *         SUBROUTINE TO FAKE AN ISR IN CASE AN INPUT FROM INTOSTAT
1021                    *             CAUSED HARDWARE TO MISS AN INTERRUPT.
1022                    *
1023                    *         ENTRY:  D1.B = INTOSTAT(A1) WHICH CAUSED INTERRUPT
1024                    *
1025                    *         HPL ROUTINE ( MODIFIED )
1026                    *
1027                    ********************************************************************************
1028  0000038C 4E4B     H_FAKEISR TRAP  #11                      GET INTO SUPERVISOR MODE        scs
1029                    * scs     MOVE    SR,-(SP)               PUT SR ON STACK FOR ISR'S RTE
1030  0000038E 48E7 FFFE          MOVEM.L D0-D7/A0-A6,-(SP)      SAVE REGISTERS
1031  00000392 832A 0034          OR.B    D1,H_INTOCOPY(A2)      PUT BYTE WHERE ISR WILL SEE IT
1032  00000396 6100 FD9A          BSR     SET_INT_LEVEL          DISABLE CARD INTRS      wuwu TM 1/19/83
1033  0000039A 4EBA 03D4          JSR     H_ISR                  CALL ISR
1034  0000039E 4CDF 7FFF          MOVEM.L (SP)+,D0-D7/A0-A6      RESTORE REGISTERS
1035  000003A2 46DF               MOVE    (SP)+,SR               RESTORE USER MODE               scs
1036  000003A4 4E75               RTS                                                            scs
1037                    * scs     RTE                            Re-enable interrupts and get SR off stack
```



```
1039                    ********************************************************************************
1040                    *
1041                    *       H_RDS
1042                    *
1043                    *         READ STATUS
1044                    *
1045                    *         PASCAL ROUTINE
1046                    *
1047                    ********************************************************************************
1048            0000 0002 H_ROUTINE EQU    2
1049            0000 0001 H_TEMP    EQU    1
1050            0000 0000 H_CRDREG  EQU    0
1051                    *
1052                    *
1053  000003A6 41FA 0072 H_RDS      LEA    H_RDSTBL,A0             get pointer to lookup table
1054  000003AA D241                ADD.W  D1,D1                   multiply the rds register by 2
1055  000003AC B23C 0012           CMP.B  #H_RT_SIZ,D1            \ check for out of bounds
1056  000003B0 6C1E                BGE.S  RDS_ERR                 /
1057  000003B2 3030 1000           MOVE.W 0(A0,D1),D0             get the table entry
1058  000003B6 6B18                BMI.S  RDS_ERR                 if the entry is 0 then error
1059  000003B8 B03C 0001           CMP.B  #H_TEMP,D0
1060  000003BC 6718                BEQ.S  HR_TEMP
1061  000003BE 6D1E                BLT.S  HR_CARD
1062  000003C0 E048                LSR    #8,D0                   get the routine offset
1063  000003C2 6726                BEQ.S  H_RDS_ID                -  status rtn 0 - card id
1064  000003C4 5340                SUBQ   #1,D0
1065  000003C6 6728                BEQ.S  H_RDS_CS                -  status 3 - ctrl status + address
1066  000003C8 5340                SUBQ   #1,D0
1067  000003CA 6736                BEQ.S  H_RDS_ST                -  status 6 - chip state
1068  000003CC 5340                SUBQ   #1,D0
1069  000003CE 673E                BEQ.S  H_RDS_LI                -  status 7 - bus lines
1070                    *           BRA.S  RDS_ERR                 there are no more status 'routines'
1071
1071
1072  000003D0 7013     RDS_ERR    MOVEQ  #BAD_RDS,D0             bad read status
1073  000003D2 6000 FE80           BRA    ESC_ERR
1074
1074
1075                    *
1076                    *         retrieve temps as words
1077                    *
1078  000003D6 E048     HR_TEMP    LSR    #8,D0                   get temp offset
1079  000003D8 3032 0000           MOVE.W 0(A2,D0),D0             get the value
1080  000003DC 4E75                RTS
1081
1081
1081
1082                    *
1083                    *         retrieve card registers as bytes
1084                    *
1085  000003DE E048     HR_CARD    LSR    #8,D0                   get the card offset
1086  000003E0 1031 0000 HR_CARD1  MOVE.B 0(A1,D0),D0             get the value
1087  000003E4 0240 00FF           ANDI.W #$00FF,D0               mask off garbage
1088  000003E8 4E75                RTS
1089
1089
1089
```

```
1090                        *
1091                        *          card id
1092                        *
1093    000003EA 303C 0001 H_RDS_ID  MOVE.W #1,D0
1094                        * tm      CMPA.L #H_INT_CA,A1          is this the internal card ?
1095                        * tm      BNE.S  HR_CARD1
1096    000003EE 4E75               RTS
1097
1098    000003F0 1029 0005 H_RDS_CS  MOVE.B H_EXTSTAT(A1),D0      get sys ctl and active ctl
1099    000003F4 0840 0006          BCHG   #6,D0                 complement NOT actv ctl
1100    000003F8 0240 00C0          ANDI.W #$C0,D0               mask bits
1101    000003FC D02A 0033          ADD.B  MA(A2),D0             get my address
1102    00000400 4E75               RTS
1103
1104    00000402 61EC      H_RDS_ST BSR    H_RDS_CS              get sys/act ctl and address info
1105    00000404 1229 0015          MOVE.B H_ADRSSTAT(A1),D1     get chip state
1106    00000408 E149               LSL    #8,D1
1107    0000040A D041               ADD.W  D1,D0                 put together in D0
1108    0000040C 4E75               RTS
1109
1110    0000040E 1029 0017 H_RDS_LI  MOVE.B H_BUSSTAT(A1),D0      get bus lines
1111    00000412 E148               LSL    #8,D0
1112    00000414 1029 001F          MOVE.B H_DATAIN(A1),D0       get data lines
1113    00000418 4E75               RTS
1114
1114
1114
1115              0000 041A H_RDSTBL  EQU    *
1116    0000041A 00                 DC.B   0,H_ROUTINE           status 0 - routine  0   - card id
1116    0000041B 02
1117    0000041C 03                 DC.B   3,H_CRDREG            status 1 - card reg 3   - intr/dma status
1117    0000041D 00
1118    0000041E 63                 DC.B   99,H_ROUTINE          status 2 - not implemented
1118    0000041F 02
1119    00000420 01                 DC.B   1,H_ROUTINE           status 3 - status&addr  - sys & act ctl my addr
1119    00000421 02
1120    00000422 34                 DC.B   H_INFOCOPY,H_TEMP     status 4 - temps
1120    00000423 01
1121    00000424 36                 DC.B   H_INTMSKSAV,H_TEMP    status 5 - temps
1121    00000425 01
1122    00000426 02                 DC.B   2,H_ROUTINE           status 6 - card reg 21+ - state
1122    00000427 02
1123    00000428 03                 DC.B   3,H_ROUTINE           status 7 - card reg 23  - bus state
1123    00000429 02
1124    0000042A 1D                 DC.B   H_CMDPASS,H_CRDREG    status 8 - card reg     - command
1124    0000042B C0
1125              0000 042C H_RT_END  EQU    *
1126              0000 0012 H_RT_SIZ  EQU    H_RT_END-H_RDSTBL    size of table
```



```
1128                        ********************************************************************************
1129                        *
1130                        *       H_WTC
1131                        *
1132                        *         WRITE CONTROL
1133                        *
1134                        *         ENTRY:  D0.W = PARAMETER
1135                        *
1136                        ********************************************************************************
1137    0000042C 0C41 0006 H_WTC     CMPI.W #6,D1                 \ check for ctl limits
1138    00000430 6C9E               BGE.S  RDS_ERR               /
1139    00000432 48C1               EXT.L  D1
1140    00000434 D281               ADD.L  D1,D1
1141    00000436 4EFB 1002          JMP    HWTCTBL(D1)
1142
1142
1143    0000043A 6012      HWTCTBL  BRA.S  H_WTC_RST             CONTROL 0 - DO A RESET
1144    0000043C 604E               BRA.S  H_RQS                 CONTROL 1 - set SRQ response
1145    0000043E 6038               BRA.S  H_WTC_PPC             CONTROL 2 : ppoll configure
1146    00000440 6010               BRA.S  H_WTC_SMA             CONTROL 3 - set my addr
1147    00000442 608C               BRA.S  RDS_ERR               CONTROL 4 : not used
1148    00000444 607E               BRA.S  H_EIR                 CONTROL 5 : enable intrpts
1149
1149
1149
1149
1150                        ********************************************************************************
1151                        *
1152                        *       HPL_WTC
1153                        *
1154                        *         WRITE CONTROL TO HP-IB ( alter my addr or set ppoll conf. )
1155                        *
1156                        *         ENTRY:  D0.W = PARAMETER
1157                        *
1158                        *         NOTE:   This command was illegal on a 9825.  On WILDFIRE, it is used
1159                        *                 to alter the HP-IB address of a given card and/or reset the
1160                        *                 card without generating IFC or locally configure the
1161                        *                 parallel poll response:
1162                        *
1163                        *                 wtc 7,20    will change address to 20 and reset card
1164                        *                 wtc 7,31    will reset the card without changing addr
1165                        *                 wtc 7,<configure byte>  will do PP configure.
1166                        *
1167                        *         HPL ROUTINE ( MODIFIED )
1168                        *
1169                        ********************************************************************************
1170    00000446 B07C 001F HPL_WTC   CMP    #31,D0                  MAKE SURE PARM IS 0..31
1171    0000044A 621E               BHI.S  HPL_WTC3                IF NOT, GO TO PPOLL CONFIGURE
1172    0000044C 6604      HPL_WTC0 BNE.S  HPL_WTC1                IF PARM <31, THEN SKIP
1173
1173
1174              0000 044E H_WTC_RST EQU    *
1175    0000044E 102A 0033          MOVE.B MA(A2),D0             ELSE USE PREVIOUS ADDRESS
1176
1176
```

```
1177            0000 0452 H_WTC_SMA EQU    *
1178    00000452 1F29 0005 HPL_WTC1  MOVE.B H_EXTSTAT(A1),-(SP)    SAVE CONTROLLER ACTIVE STATE.
1179    00000456 6100 FD34          BSR    H_INIT_S               DO SOFTWARE RESET
1180    0000045A 081F 0006          BTST   #6,(SP)+               WERE WE CONTROLLER?
1181    0000045E 6608               BNE.S  HPL_WTC2               IF NOT, SKIP
1182    00000460 10BC 0011          MOVE.B #H_RQC,(A0)            ELSE REGAIN CONTROL
1183    00000464 10BC 000B          MOVE.B #H_GTS,(A0)            AND RE-DROP ATN
1184    00000468 4E75       HPL_WTC2 RTS                          ELSE DONE
1185
1185
1185
1186    0000046A 7200       HPL_WTC3 MOVEQ  #0,D1                 COMPUTE THE TWO BYTE PPOLLMSK
1187    0000046C 01C1               BSET   D0,D1                  BASED ON CONFIGURATION IN D0.
1188    0000046E 3541 003A          MOVE.W D1,H_PPOLLMSK(A2)      SAVE THE MASK
1189    00000472 6100 0038          BSR    HPL_WTC4               set response in 9914  ( SPR695 TM 4/21/82 )
1190    00000476 4E75               RTS                                                 ( SPR695 TM 4/21/82 )
1191
1191
1191
1192            0000 0478 H_WTC_PPC EQU    *
1193    00000478 1200               MOVE.B D0,D1                  copy so HPL_WTC4 works( 564    TM 10/6/82 )
1194    0000047A E149               LSL.W  #8,D1                  \ duplicate in left   ( 564    TM 10/6/82 )
1195    0000047C 8200               OR.B   D0,D1                  /  byte for rsv stuff ( 564    TM 10/6/82 )
1196    0000047E 426A 003A          CLR.W  H_PPOLLMSK(A2)         clear the mask        ( SPR695 TM 4/21/82 )
1197    00000482 3541 003A          MOVE.W D1,H_PPOLLMSK(A2)      move new mask in temps( SPR695 TM 4/21/82 )
1198    00000486 6100 0024          BSR    HPL_WTC4               set response in 9914  ( SPR695 TM 4/21/82 )
1199    0000048A 4E75               RTS
1200
1200
1200                        ********************************************************************************
1201                        *
1202                        *       H_RQS
1203                        *
1204                        *         request service - set spoll response ( & SRQ )
1205                        *
1206                        *         NOTE : rsv is the Request SerVice bit.
1207                        *                On the 9825 this is tied to the
1208                        *                ist state ( individual status ).
1209                        *                In the IEEE 488 standard the rsv state is
1210                        *                the SRQ response and ist is the  PPOLL
1211                        *                response.  The standard does not
1212                        *                specify any relation between the
1213                        *                two.
1214                        *
1215                        *         PASCAL ROUTINE ( taken from HPL R70UT )
1216                        *
1217                        ********************************************************************************
1218    0000048C 08AA 0006 H_RQS     BCLR   #6,H_FLAGS(A2)         ASSUME rsv = 0 IN THIS NEW BYTE
                0039
1219    00000492 1200               MOVE.B D0,D1                  IF rsv BIT IN THE NEW BYTE IS INDEED
1220    00000494 0881 0006          BCLR   #6,D1                  ZERO, THEN JUST OUTPUT THE NEW BYTE.
1221    00000498 670A               BEQ.S  H_RQS2
1222    0000049A 1341 001B          MOVE.B D1,H_SPOLL(A1)         ELSE FIRST WRITE THE BYTE WITH rsv
1223    0000049E 08EA 0006          BSET   #6,H_FLAGS(A2)         CLEAR. REMEMBER THAT rsv IS SET.
                0039
1224    000004A4 1340 001B H_RQS2    MOVE.B D0,H_SPOLL(A1)         WRITE THE BYTE WITH rsv CORRECT.
```



```
1226    000004A8 322A 003A          MOVE.W H_PPOLLMSK(A2),D1      GO UPDATE THE PARALLEL POLL RESPONSE
1227    000004AC 4E4B       HPL_WTC4 TRAP  #11                                              scs
1228                        * scs    MOVE   SR,-(SP)               DISABLE ISR'S WHILE WE FIGURE
1229    000004AE 007C 2700          ORI    #$2700,SR              OUT WHICH MASK TO SET BASED ON
1230    000004B2 082A 0006          BTST   #6,H_FLAGS(A2)         CURRENT rsv BIT.
                0039
1231    000004B8 6702               BEQ.S  HPL_WTC5               IF rsv = 0, USE RIGHT BYTE
1232    000004BA E059               ROR    #8,D1                  ELSE USE LEFT BYTE
1233    000004BC 1341 001D HPL_WTC5 MOVE.B D1,H_PPOLL(A1)
1234    000004C0 46DF               MOVE   (SP)+,SR                                         scs
1235    000004C2 4E75               RTS                                                     scs
1236                        * scs    RTE                           RE-ENABLE ISR'S AND RETURN
```

```
1238                        ********************************************************************************
1239                        *
1240                        *       H_EIR
1241                        *
1242                        *         ENABLE THE HP-IB BASED ON 98034 ENABLE BYTE
1243                        *
1244                        *         ENTRY:  D0.B = EIR BYTE A LA 98034 CARD
1245                        *
1246                        *         EXIT:   EIR BYTE SAVED IN DRIVER TEMPS.
1247                        *                 EIR ACCOMPLISHED.  ANY CONDITIONS ALREADY TRUE GENERATE LOGIN.
1248                        *
1249                        *         USES:   D0, D1, D2
1250                        *
1251                        *         HPL ROUTINE
1252                        *
1253                        ********************************************************************************
1254  000004C4 1540 002C H_EIR      MOVE.B D0,EIRB_OFF(A2)          SAVE EIR BYTE
1255  000004C8 137C 0093            MOVE.B #H_DAI1,H_AUXCMD(A1)     DISABLE ALL INTS FOR A SEC
                 0017
1256                        * tm     MOVE.B #$80,3(A1)               ENABLE THE CARD       ( hphp TM 1/19/83 )
1257  000004CE 323C CEAB            MOVE.W #$CEAB,D1                D1 = INITIAL VALUE OF INTMSK
1258  000004D2 3400                 MOVE.W D0,D2                    EXTRACT IRF & ORE BITS FROM BYTE
1259  000004D4 C47C 000C            AND    #$C,D2
1260  000004D8 E14A                 LSL    #8,D2                    MOVE THESE BITS TO BI/BO POSITION
1261  000004DA E54A                 LSL    #2,D2
1262  000004DC 8242                 OR     D2,D1                    AND INCLUDE IN THE ENABLE MASK
1263  000004DE 3400                 MOVE.W D0,D2                    SAVE EIR BYTE IN D2 WHILE LOOKING FOR
1264  000004E0 C07C 0030            AND    #$30,D0                  EITHER TLK OR LST BITS ON?
1265  000004E4 6704                 BEQ.S  H_EIR2                   IF NOT, SKIP
1266  000004E6 D27C 0004              ADD  #4,D1                     ELSE ENABLE MA TO INTERRUPT
1267  000004EA 3541 0036 H_EIR2     MOVE.W D1,H_INTMSKSAV(A2)       SAVE THIS MASK VALUE
1268  000004EE 0389 0011            MOVEP  D1,H_INTOMASK(A1)        GIVE MASK TO 9914
1269  000004F2 102A 0039            MOVE.B H_FLAGS(A2),D0           GENERATE IMMEDIATE INTERRUPT IF ANY
1270  000004F6 C07C 003C            AND    #$3C,D0                  OF THE 'OTHER' CONDITIONS ARE TRUE
1271  000004FA 6706                  BEQ.S H_EIR3
1272  000004FC 4EB9 0000            JSR    LOGINT
                 0000
1273  00000502 6100 0570 H_EIR3     BSR    H_CHKADDR                GENERATE ANY ADDRESS INTERRUPTS.
1274  00000506 6100 058E            BSR    H_CHKSRQ                 GENERATE SRQ INTERRUPT IF NECESSARY
1275  0000050A 122A 0034            MOVE.B H_INTOCOPY(A2),D1        IF BI/BO IS ENABLED AND THE
1276  0000050E C22A 0036            AND.B  H_INTMSKSAV(A2),D1       BO/BI STATUS IS ALREADY TRUE,
1277  00000512 C23C 0030            AND.B  #$30,D1                  THEN WE HAVE TO FAKE AN
1278  00000516 6704                 BEQ.S  H_EIR4                   INTERRUPT.
1279  00000518 6100 FE72              BSR  H_FAKEISR
1280  0000051C 137C 0013 H_EIR4     MOVE.B #H_DAI0,H_AUXCMD(A1)     RENABLE ALL INTS FOR CARD
                 0017
1281  00000522 08AA 0001            BCLR    #1,H_FLAGS(A2)          if isr pend then do it ( rrrr TM 12/17/82 )
                 0039
1282  00000528 6706                 BEQ.S   H_EIR5                     else just exit       ( rrrr TM 12/17/82 )
1283  0000052A 4EB9 0000            JSR     LOGINT                                          ( rrrr TM 12/17/82 )
                 0000
1284  00000530 4E75       H_EIR5    RTS                                                     ( rrrr TM 12/17/82 )
```



```
1286                        ********************************************************************************
1287                        *
1288                        *       H_ENABLE
1289                        *
1290                        *         ENABLE THE HP-IB FOR TRANSFER USES
1291                        *
1292                        *         ENTRY:  D0.W = 9914 ENABLE MASK ( TO BE INCLUSIVE OR'ED )
1293                        *                 ( D0 = #$2000  FOR BYTE IN  ( BI )
1294                        *                 ( D0 = #$1000  FOR BYTE OUT ( BO )
1295                        *
1296                        *         USES:   D0, D1
1297                        *
1298                        *         PASCAL ROUTINE        1/19/83
1299                        *
1300                        ********************************************************************************
1301             0000 0532 H_ENABLE  EQU   *
1302  00000532 137C 0093            MOVE.B  #H_DAI1,H_AUXCMD(A1)  DISABLE THE CARD     ( hphp TM 1/19/83 )
                 0017
1303  00000538 806A 0036            OR.W    H_INTMSKSAV(A2),D0    OR NEW BITS IN       ( hphp TM 1/19/83 )
1304  0000053C 3540 0036 H_ED_COM   MOVE.W  D0,H_INTMSKSAV(A2)    AND RE-SAVE          ( hphp TM 1/19/83 )
1305  00000540 0189 0011            MOVEP   D0,H_INTOMASK(A1)     AND GIVE TO CARD     ( hphp TM 1/19/83 )
1306  00000544 122A 0034            MOVE.B  H_INTOCOPY(A2),D1     IF BI/BO ENABLED &   ( hphp TM 1/19/83 )
1307  00000548 C22A 0036            AND.B   H_INTMSKSAV(A2),D1       IS ALREADY TRUE,  ( hphp TM 1/19/83 )
1308  0000054C C23C 0030            AND.B   #$30,D1               THEN FAKE AN         ( hphp TM 1/19/83 )
1309  00000550 6704                 BEQ.S   H_ED_EXIT             INTERRUPT.           ( hphp TM 1/19/83 )
1310  00000552 6100 FE38            BSR     H_FAKEISR                                  ( hphp TM 1/19/83 )
1311  00000556 137C 0013 H_ED_EXIT  MOVE.B  #H_DAI0,H_AUXCMD(A1)   RENABLE CARD        ( hphp TM 1/19/83 )
                 0017
1312  0000055C 4E75                 RTS                                                ( hphp TM 1/19/83 )
1313                        ********************************************************************************
1314                        *
1315                        *       H_DISABLE
1316                        *
1317                        *         DISABLES BO AND BI ON THE HP-IB FOR TRANSFER USES
1318                        *
1319                        *         USES:   D0, D1
1320                        *
1321                        *         PASCAL ROUTINE        1/19/83
1322                        *
1323                        ********************************************************************************
1324             0000 055E H_DISABLE EQU   *
1325  0000055E 137C 0093            MOVE.B  #H_DAI1,H_AUXCMD(A1)  DISABLE THE CARD     ( hphp TM 1/19/83 )
                 0017
1326  00000564 302A 0036            MOVE.W  H_INTMSKSAV(A2),D0    GET OLD MASK         ( hphp TM 1/19/83 )
1327  00000568 0240 CFFF            ANDI.W  #$CFFF,D0             MASK OUT BO/BI       ( hphp TM 1/25/83 )
1328  0000056C 60CE                 BRA.S   H_ED_COM              JUMP TO COMMON CODE  ( hphp TM 1/19/83 )
```

```
1330                        ********************************************************************************
1331                        *
1332                        *       H_P_POLL
1333                        *
1334                        *         CONDUCT PARALLEL POLL
1335                        *
1336                        *         IF NOT ACTIVE CONTROLLER GIVE ERROR
1337                        *         ELSE VALUE RETURNED IN D0.B
1338                        *
1339                        *         HPL ROUTINE
1340                        *
1341                        ********************************************************************************
1342    0000056E 6100 0092 H_P_POLL  BSR     H_SET_ATN              SET ATN LINE
1343    00000572 6100 FCF6           BSR     H_WAIT_BO              WAIT FOR 'READY'
1344    00000576 002A 0010           ORI.B   #16,H_INTOCOPY(A2)     (SAVE BO STATUS FOR LATER)
                 0034
1345    0000057C 137C 008E           MOVE.B #H_RPP1,H_AUXCMD(A1)    REQUEST THE PARALLEL POLL
                 0017
1346                        *   JS     MOVEQ  #20,D0                DELAY 40 US FOR LINES TO SETTLE
1347                        *   JS     DBRA   D0,*
1348    00000582 2F3C 0000           MOVE.L #40,-(SP)               USE TIMER FOR DELAY   tttt JS 8/1/83
                 0028
1349    00000588 4EB9 0000           JSR     DELAY_TIMER                                  tttt JS 8/1/83
                 0000
1350    0000058E 7000                MOVEQ  #0,D0
1351    00000590 1029 001D           MOVE.B H_CMDPASS(A1),D0        GET THE RESPONSE
1352    00000594 137C 000E           MOVE.B #H_RPP0,H_AUXCMD(A1)    CLEAR PARALLEL POLL
                 0017
1353    0000059A 137C 000B           MOVE.B #H_GTS,H_AUXCMD(A1)     CLEAR ATN
                 0017
1354    000005A0 4E75                RTS
```



```
1356                        ********************************************************************************
1357                        *
1358                        *       H_SET
1359                        *
1360                        *         Set an HPIB line true
1361                        *
1362                        *         PASCAL ROUTINE
1363                        *
1364                        ********************************************************************************
1365    000005A2 B27C 0007 H_SET     CMP     #7,D1                  \
1366    000005A6 6200 FC8A           BHI     H_SC_ERR               /  make sure bit # is  <=7
1367    000005AA D241                ADD     D1,D1
1368    000005AC 41FA 0008           LEA     H_S_TBL,A0             \
1369    000005B0 D0F0 1000           ADDA.W 0(A0,D1),A0             \INDEXED JUMP THRU TABLE
1370    000005B4 4ED0                JMP     (A0)                   /
1371                        *                               move h_s_tbl(d1),d1
1372                        *                               jmp  h_s_tbl(d1)
1373
1373
1373
1374            0000 05B6 H_S_TBL    EQU     *
1375    000005B6 0010                DC.W    H_REN-H_S_TBL          REN  - set REN
1376    000005B8 001C                DC.W    H_IFC-H_S_TBL          IFC  - pulse IFC ( set REN/clr ATN )
1377    000005BA FC7C                DC.W    H_SC_ERR-H_S_TBL       SRQ  - error
1378    000005BC 0084                DC.W    H_EOI-H_S_TBL          EOI  - pulse EOI on next byte out
1379    000005BE FC7C                DC.W    H_SC_ERR-H_S_TBL       NRFD - error
1380    000005C0 FC7C                DC.W    H_SC_ERR-H_S_TBL       NDAC - error
1381    000005C2 FC7C                DC.W    H_SC_ERR-H_S_TBL       DAV  - error
1382    000005C4 004C                DC.W    H_SET_ATN-H_S_TBL      ATN  - set ATN true
1383
1383
1383
1384                        ********************************************************************************
1385                        *
1386                        *       H_REN
1387                        *
1388                        *         SET REN ON HP-IB
1389                        *
1390                        *         EXIT :   IF NOT SYSTEM CONTROLLER THEN GIVE ERROR
1391                        *
1392                        *         HPL ROUTINE
1393                        *
1394                        ********************************************************************************
1395    000005C6 0829 0007 H_REN     BTST    #7,H_EXTSTAT(A1)
                 0005
1396    000005CC 6700 FC6C           BEQ     H_NOTSCTL
1397    000005D0 6028                BRA.S   H_IFC2
1398
1398
1398
1399                        ********************************************************************************
1400                        *
1401                        *       H_IFC
1402                        *
1403                        *         DRIVE IFC TRUE FOR 100 MICROSECONDS
1404                        *
1405                        *         ENTRY :   IF NOT SYSTEM CONTROLLER, CLEAR STS AND SET ERR
```

```
1406                            *
1407                            *         EXIT  :   ATN CLEARED
1408                            *                   REN SET
1409                            *
1410                            *         NOTE  :   IF THE 9914 IS NOT IN SOFTWARE RESET, THIS ROUTINE WILL
1411                            *                   DRIVE THE ATN LINE TRUE DURING THE IFC.
1412                            *
1413                            *         HPL ROUTINE
1414                            *
1415                            *
1416                            ********************************************************************************
1417  000005D2 0829 0007 H_IFC      BTST   #7,H_EXTSTAT(A1)       MUST BE SYSTEM CONTROLLER
                 0005
1418  000005D8 6700 FC60            BEQ    H_NOTSCTL
1419  000005DC 137C 008F            MOVE.B #H_SIC1,H_AUXCMD(A1)   SET IFC
                 0017
1420                            *   JS     MOVEQ  #70,D0                 SET DELAY COUNT
1421                            *   JS     DBRA   D0,*
1422  000005E2 2F3C 0000            MOVE.L #100,-(SP)             USE TIMER FOR DELAY  tttt JS 8/1/83
                 0064
1423  000005E8 4EB9 0000            JSR    DELAY_TIMER                                tttt JS 8/1/83
                 0000
1424  000005EE 137C 000F            MOVE.B #H_SIC0,H_AUXCMD(A1)   CLEAR IFC
                 0017
1425  000005F4 137C 000B            MOVE.B #H_GTS,H_AUXCMD(A1)    CLEAR ATN
                 0017
1426  000005FA 137C 0090 H_IFC2     MOVE.B #H_SRE1,H_AUXCMD(A1)   SET REN
                 0017
1427  00000600 4E75                 RTS
1428
1428
1428
1429                            ********************************************************************************
1430                            *
1431                            *       H_SET_ATN
1432                            *
1433                            *         ROUTINE TO SET ATN.
1434                            *
1435                            *         EXIT:   IF NOT CONTROLLER THEN GIVE ERROR
1436                            *                    ELSE IF ADDRESSED TO LISTEN AND HOLDOFF FLAG IS SET
1437                            *                    THEN DO TCS
1438                            *                 ELSE IF ADDRESSED TO TALK,
1439                            *                    THEN WAIT FOR BO STATUS AND DO TCA
1440                            *                 ELSE DO TCA
1441                            *
1442                            *         USES:   H_WAIT ROUTINE
1443                            *
1444                            *         HPL ROUTINE ( H_SET_ATN used to be H_ATN1 )
1445                            *                     ( H_SET_ATN was identical to H_ATN1
1446                            *                       except for error exits )
1447                            *
1448                            ********************************************************************************
1449  00000602 0829 0006 H_SET_ATN BTST   #6,H_EXTSTAT(A1)       BETTER BE CONTROLLER
                 0005
1450  00000608 6600 FC2C            BNE    H_NOTACTL             ELSE ERROR
1451  0000060C 1229 0015 H_ATN1A    MOVE.B H_ADRSSTAT(A1),D1     GET ADDRESSED STATUS
1452  00000610 740D                 MOVEQ  #H_TCS,D2             ASSUME WE CAN DO TCS
```



```
1453  00000612 0801 0002            BTST   #2,D1                 ARE WE A LISTENER?
1454  00000616 670A                 BEQ.S  H_ATN1_0              IF NOT, SKIP
1455  00000618 082A 0007            BTST   #7,H_FLAGS(A2)        TEST HOLDOFF FLAG
                 0039
1456  0000061E 6614                 BNE.S  H_ATN1_3              IF IT WAS SET, USE THE TCS
1457  00000620 6010                 BRA.S  H_ATN1_2              ELSE DO TCA
1458  00000622 0801 0001 H_ATN1_0   BTST   #1,D1                 ARE WE A TALKER?
1459  00000626 670A                 BEQ.S  H_ATN1_2              IF NOT, TAKE CONTROL ASYNC
1460  00000628 6100 FC40 H_ATN1_1   BSR    H_WAIT_BO             ELSE WAIT FOR BYTE OUT
1461  0000062C 002A 0010            ORI.B  #I6,H_INTOCOPY(A2)    (SAVE BO STATUS FOR LATER!)
                 0034
1462  00000632 740C      H_ATN1_2   MOVEQ  #H_TCA,D2             DO TAKE CONTROL ASYNC
1463  00000634 1342 0017 H_ATN1_3   MOVE.B D2,H_AUXCMD(A1)       TAKE CONTROL!
1464  00000638 4E75                 RTS
1465
1465
1465
1466                            ********************************************************************************
1467                            *
1468                            *       H_EOI
1469                            *
1470                            *         ROUTINE TO SET EOI ON THE NEXT BYTE OUT
1471                            *
1472                            *         test to see if 9914 waits -
1473                            *               if so - ok
1474                            *               if not- do wait ( HW... )
1475                            *
1476                            ********************************************************************************
1477  0000063A 137C 0008 H_EOI      MOVE.B #H_FEOI,H_AUXCMD(A1)   SET EIO WITH NEXT BYTE
                 0017
1478  00000640 4E75      H_DMYRTS   RTS
```

```
1480                        ********************************************************************************
1481                        *
1482                        *       H_CLR
1483                        *
1484                        *         Set an HPIB line false
1485                        *
1486                        *         PASCAL ROUTINE
1487                        *
1488                        ********************************************************************************
1489    00000642 B27C 0007 H_CLR      CMP     #7,D1                   \
1490    00000646 6200 FBEA            BHI     H_SC_ERR                /  make sure bit # is  <=7
1491    0000064A D241                 ADD.W   D1,D1
1492    0000064C 41FA 0008            LEA     H_C_TBL,A0              \
1493    00000650 D0F0 1000            ADDA.W  0(A0,D1),A0              INDEXED JUMP THRU TABLE
1494    00000654 4ED0                 JMP     (A0)                    /
1495
1495
1495
1496
1497            0000 0656 H_C_TBL   EQU     *
1498    00000656 0010               DC.W    H_LOCAL-H_C_TBL         REN  - clear REN
1499    00000658 FFEA               DC.W    H_DMYRTS-H_C_TBL        IFC  - nothing
1500    0000065A FBDC               DC.W    H_SC_ERR-H_C_TBL        SRQ  - error
1501    0000065C FFEA               DC.W    H_DMYRTS-H_C_TBL        EOI  - nothing
1502    0000065E FBDC               DC.W    H_SC_ERR-H_C_TBL        NRFD - error
1503    00000660 FBDC               DC.W    H_SC_ERR-H_C_TBL        NDAC - error
1504    00000662 FBDC               DC.W    H_SC_ERR-H_C_TBL        DAV  - error
1505    00000664 0022               DC.W    H_CLR_ATN-H_C_TBL       ATN  - clear ATN
1506
1506
1506
1507                        ********************************************************************************
1508                        *
1509                        *       H_LOCAL
1510                        *
1511                        *         CLEAR REN ON HP-IB
1512                        *
1513                        *         EXIT :   IF NOT SYSTEM CONTROLLER THEN GIVE ERROR
1514                        *
1515                        *         PASCAL ROUTINE
1516                        *
1517                        ********************************************************************************
1518    00000666 0829 0007 H_LOCAL    BTST    #7,H_EXTSTAT(A1)
                0005
1519    0000066C 6700 FBCC            BEQ     H_NOTSCTL
1520    00000670 137C 0010            MOVE.B #H_SRE0,H_AUXCMD(A1)   CLEAR REN
                0017
1521    00000676 4E75                 RTS
1522
1522
1522
1523                        ********************************************************************************
1524                        *
1525                        *       H_CLR_ATN
1526                        *
1527                        *         CLEAR ATN ON HP-IB
1528                        *
```



```
1529                        *         EXIT :   IF NOT ACTIVE CONTROLLER THEN GIVE ERROR
1530                        *
1531                        *         PASCAL ROUTINE
1532                        *
1533                        ********************************************************************************
1534    00000678 0829 0006 H_CLR_ATN BTST   #6,H_EXTSTAT(A1)
                0005
1535    0000067E 6600 FBB6            BNE     H_NOTACTL
1536    00000682 137C 000B            MOVE.B #H_GTS,H_AUXCMD(A1)    CLEAR ATN
                0017
1537    00000688 4E75                 RTS
1538
1538
1538
```

```
1540                        ********************************************************************************
1541                        *
1542                        *       H_TEST
1543                        *
1544                        *         Get an HPIB line's state
1545                        *
1546                        *         ENTRY :       D1 = line parameter
1547                        *                           0 = REN
1548                        *                           1 = IFC
1549                        *                           2 = SRQ
1550                        *                           3 = EOI
1551                        *                           4 = NRFD
1552                        *                           5 = NDAC
1553                        *                           6 = DAV
1554                        *                           7 = ATN
1555                        *
1556                        *         PASCAL ROUTINE
1557                        *
1558                        ********************************************************************************
1559    0000068A 7001       H_TEST    MOVEQ #1,D0                     assume line is true
1560    0000068C 0329 0017            BTST   D1,H_BUSSTAT(A1)         test bus lines from 9914
1561    00000690 6602                 BNE.S  H_TEST_EX                if line is set then return
1562    00000692 4240                 CLR.W  D0                       else set false return
1563    00000694 4E75       H_TEST_EX RTS
```



```
1565                        ********************************************************************************
1566                        *
1567                        *       H_R6OUT
1568                        *
1569                        *         EMULATION OF R6 OUT FOR HP-IB
1570                        *
1571                        *         ENTRY:  D0 = BYTE TO OUTPUT
1572                        *
1573                        *         EXIT:   IF NOT ACTIVE CONTROLLER, STS CLEARED AND ERROR BIT SET
1574                        *                 ELSE OPERATION IS DONE AND ANY ADDRESSING DECODED.
1575                        *
1576                        *         HPL ROUTINE ( MODIFIED )
1577                        *
1578                        ********************************************************************************
1579    00000696 4EB9 0000 H_R6OUT   JSR    WAIT_TFR                IF A TFR IS ACTIVE WAIT TILL IT ISN'T
                    0000
1580    0000069C 6100 FF64            BSR    H_SET_ATN               GO SET ATN OR GIVE STS ERROR
1581    000006A0 08EA 0000            BSET   #0,H_FLAGS(A2)          SET PASS CONTROL FLAG
                    0039
1582    000006A6 6100 FB7C            BSR    H_WTB1                  GO OUTPUT THE BYTE
1583    000006AA C07C 007F            AND    #$7F,D0                 CLEAR MSB FOR COMPARISON
1584    000006AE 142A 0033            MOVE.B MA(A2),D2               GET MY ADDRESS
1585    000006B2 847C 0020            OR     #$20,D2                 MAKE A LISTEN ADDRESS
1586                        * tm      MOVEQ  #H_LON1,D1              ASSUME LON COMMAND...
1587    000006B6 123C 0089            MOVE.B #H_LON1,D1
1588    000006BA B002                 CMP.B  D2,D0                   MLA?
1589    000006BC 6726                 BEQ.S  H_R6OUT3                IF SO GO DO LON
1590    000006BE 0A42 0060            EORI   #$60,D2                 MAKE A TALK ADDRESS
1591                        * tm      MOVEQ  #H_TON1,D1              ASSUME TON COMMAND...
1592    000006C2 123C 008A            MOVE.B #H_TON1,D1
1593    000006C6 B002                 CMP.B  D2,D0                   MTA?
1594    000006C8 671A                 BEQ.S  H_R6OUT3                IF SO, GO DO TON
1595    000006CA 7209                 MOVEQ  #H_LON0,D1              ASSUME LON0 COMMAND
1596    000006CC B03C 003F            CMP.B  #UNL,D0                 UNLISTEN?
1597    000006D0 6712                 BEQ.S  H_R6OUT3                IF SO, GO DO LON0
1598    000006D2 B03C 0009            CMP.B  #TCT,D0                 TAKE CONTROL?
1599    000006D6 6718                 BEQ.S  H_R6TCT                 IF SO , SKIP to tct code
1600    000006D8 C07C 0060 H_R6OUT2   AND    #$60,D0                 OTHER TALK ADDRESS?
1601    000006DC B03C 0040            CMP.B  #$40,D0
1602    000006E0 6606                 BNE.S  H_R6OUT4                IF NOT, SKIP
1603    000006E2 720A                 MOVEQ  #H_TON0,D1              IF SO, SET UP FOR TON0
1604    000006E4 1341 0017 H_R6OUT3   MOVE.B D1,H_AUXCMD(A1)         IF SO, DO IT
1605    000006E8 08AA 0000 H_R6OUT4   BCLR   #0,H_FLAGS(A2)          CLEAR PASS CONTROL FLAG
                    0039
1606    000006EE 4E75                 RTS
1607
1607
1607
1608    000006F0 0829 0001 H_R6TCT    BTST   #1,H_ADRSSTAT(A1)       ARE WE TALKER?
                    0015
1609    000006F6 66F0                 BNE.S  H_R6OUT4                IF SO, IGNORE TCT
1610    000006F8 6100 FB70            BSR    H_WAIT_B0               IF NOT, THEN WAIT FOR BYTE OUT
1611    000006FC 137C 000B            MOVE.B #H_GTS,H_AUXCMD(A1)     AND drop atn
                    0017
1612    00000702 137C 0012            MOVE.B #H_RLC,H_AUXCMD(A1)     AND RELEASE CONTROL
                    0017
```

```
1613    00000703 0829 0006 0005 HTCTLOOP  BTST   #6,H_EXTSTAT(A1)  (tm) \
1614    0000070E 67F8           BEQ      HTCTLOOP                 (tm) / wait for non active ctl
1615    00000710 4E75           RTS                               (tm) exit
```



```
1617                    ********************************************************************************
1618                    *
1619                    *       H_DMATERM
1620                    *
1621                    *         TERMINATION OF DMA TRANSFER
1622                    *
1623                    *         CALLED FROM DMA INTERRUPT SERVICE ROUTINE
1624                    *
1625                    *         DMA RESOURCE HAS ALREADY BEEN RELEASED
1626                    *
1627                    *         HPL ROUTINE ( MODIFIED )
1628                    *
1629                    ********************************************************************************
1630    00000712 6100 FA1E H_DMATERM BSR      SET_INT_LEVEL          DISABLE CARD INTRS      wuwu TM 1/19/83
1631                    *                                          this is okay - only called in ISR
1632    00000716 4EB9 0000          JSR      DROPDMA                RELINQUISH DMA RESOURCE
                 0000
1633    0000071C 137C 0080          MOVE.B   #$80,3(A1)             TURN OFF DMA ENABLE BITS( hphp TM 1/27/83 )
                 0003
1634                    *                                          in the card.            ( hphp TM 1/27/83 )
1635    00000722 4EB9 0000          JSR      ITXFR                  MAKE SURE THERE IS A TRANSFER ACTIVE
                 0000
1636    00000728 6744               BEQ.S    HDMA_END               IF NOT, FORGET THE INTERRUPT
1637                    *                                          at this point
1638                    *                                          D4 has remaining count
1639                    *                                          D3 has intended
1640    0000072A 2744 0010          MOVE.L   D4,TCNT_OFF(A3)        update count
1641    0000072E 9684               SUB.L    D4,D3                 put # bytes tfr'd into D3
1642    00000730 4A2B 000D          TST.B    TDIR_OFF(A3)          WHAT DIRECTION OF TRANSFER?
1643    00000734 6622               BNE.S    H_DMATO                IF OUTPUT, SKIP
1644    00000736 D7AB 0020          ADD.L    D3,TFIL_OFF(A3)       update fill pointer
1645    0000073A 137C 0004          MOVE.B   #H_HDFE0,H_AUXCMD(A1) ELSE CLEAR HOLD OFF ON END MODE
                 0017
1646    00000740 137C 0083          MOVE.B   #H_HDFA1,H_AUXCMD(A1) SET HOLD OFF ON ALL MODE
                 0017
1647    00000746 303C 2000          MOVE.W   #$2000,D0              PRESET ENABLE FOR BI    ( hphp TM 1/19/83 )
1648    0000074A 277C 0000 H_DMATI_1 MOVE.L  #1,TCNT_OFF(A3)        TRANSFER LAST BYTE UNDER INTERRUPT
                 0001 0010
1649    00000752 6100 FDDE          BSR      H_ENABLE                                       ( hphp TM 1/19/83 )
1650    00000756 6016               BRA.S    HDMA_END
1651
1651
1652    00000758 D7AB 001C H_DMATO   ADD.L    D3,TEMP_OFF(A3)        update empty pointer
1653    0000075C 4A2B 0008          TST.B    TEND_OFF(A3)           IS EOI TAG SET?
1654    00000760 6706               BEQ.S    H_DMATO_1              IF NOT, TRANSFER IS DONE!
1655    00000762 303C 1000          MOVE.W   #$1000,D0              SEND LAST BYTE BY INTR  ( hphp TM 1/19/83 )
1656    00000766 60E2               BRA.S    H_DMATI_1
1657    00000768 4EB9 0000 H_DMATO_1 JSR     STCLR                  CLEAR BUFFER BUSY BITS & LOG BRANCH
                 0000
1658    0000076E 4E75       HDMA_END RTS                           END OF SERVICE
```

```
1661                        ********************************************************************************
1662                        *
1663                        *       H_ISR
1664                        *
1665                        *         INTERRUPT SERVICE ROUTINE FOR HP-IB CARDS
1666                        *
1667                        *         ENTRY : A1,A2 are set up
1668                        *
1669                        *         The ISR will track down the buffer control block
1670                        *
1671                        *         HPL ROUTINE ( MODIFIED )
1672                        *
1673                        ********************************************************************************
1674             0000 0770 H_ISR     EQU     *
1675     00000770 08AA 0001          BCLR    #1,H_FLAGS(A2)          clear user isr pending flag
             0039
1676     00000776 0109 0011          MOVEP   H_INTOSTAT(A1),D0       GET BOTH STATUS BYTES FROM 9914
1677     0000077A 6100 01EE          BSR     H_ISR1                  PROCESS THE INTERRUPTING CONDITION(S)
1678     0000077E 4EB9 0000          JSR     ITXFR                   GO SEE IF TRANSFER IS ACTIVE
             0000
1679     00000784 6714               BEQ.S   HISR_END                IF NOT, THEN WE ARE DONE.
1680     00000786 B23C 0003          CMP.B   #TT_BURST,D1            IF FRW IS ACTIVE, GO PROCESS IT
1681     0000078A 6700 010E          BEQ     H_FRW
1682     0000078E B23C 0001          CMP.B   #TT_INT,D1              IF INT THEN GO PROCESS IT
1683     00000792 677E               BEQ.S   H_BUF
1684     00000794 B23C 0002          CMP.B   #TT_DMA,D1              IF DMA
1685     00000798 6710               BEQ.S   H_ISRDMA
1686     0000079A 08AA 0001 HISR_END BCLR    #1,H_FLAGS(A2)          if isr pending then do it
             0039
1687     000007A0 6706               BEQ.S   H_ISR_EX
1688     000007A2 4EB9 0000          JSR     LOGINT
             0000
1689     000007A8 4E75       H_ISR_EX RTS                            otherwise return ( used for FAKEISR )
1690
1691                        *
1692                        * DMA TRANSFER CLEANUP:
1693                        *
1694     000007AA 0C83 0000 H_ISRDMA CMPI.L  #1,D3                   IF COUNT IS = one  ,  THIS IS
             0001
1695     000007B0 6760               BEQ.S   H_BUF                      THE EXTRA TFR BY INTERRUPT, SO SKIP
1696     000007B2 4A2B 000D          TST.B   TDIR_OFF(A3)            \
1697     000007B6 66E2               BNE.S   HISR_END                / IF OUTPUT, CAN'T BE EARLY TERM
1698     000007B8 4A2B 000B          TST.B   TEND_OFF(A3)            \
1699     000007BC 67DC               BEQ.S   HISR_END                / IF NO EOI TAG, CAN'T BE EARLY TERM
1700     000007BE 0800 000B          BTST    #11,D0                  \
1701     000007C2 67D6               BEQ     HISR_END                / IF EOI NOT SET, CAN'T BE EARLY TERM
1702     000007C4 137C 0080          MOVE.B  #$80,3(A1)              ELSE IT IS EARLY DMA TERMINATION
             0003
1703     000007CA 137C 0083          MOVE.B  #H_HDFA1,H_AUXCMD(A1) SO DISABLE DMA, SET HOLD OFF ON
             0017
1704     000007D0 137C 0004          MOVE.B  #H_HDFE0,H_AUXCMD(A1) ALL, CLEAR HOLD OFF ON END.
             0017
1705     000007D6 08EA 0007          BSET    #7,H_FLAGS(A2)          SET HOLDOFF INDICATOR
             0039
1706             0000 07DC H_BYTTST EQU     *                                       ( SPRxxx TM 6/14/82 )
```



```
1707     000007DC 082A 0005          BTST    #13-8,H_INTOCOPY(A2)  test for byte in        ( SPRxxx TM 7/21/82 )
             0034
1708     000007E2 6716               BEQ.S   H_NOBYTE                   if no byte - norm. ( SPRxxx TM 6/14/82 )
1709     000007E4 4EB9 0000          JSR     DROPDMA               free the dma channel    ( SPRxxx TM 6/14/82 )
             0000
1710                        *                                         D4 has remaining = 0    ( SPRxxx TM 6/14/82 )
1711                        *                                         D3 has intended         ( SPRxxx TM 6/14/82 )
1712     000007EA 303C 2000          MOVE.W  #$2000,D0               set bit #13 - byte in
1713     000007EE D7AB 0020          ADD.L   D3,TFIL_OFF(A3)         update fill pointer     ( SPRxxx TM 6/14/82 )
1714     000007F2 206B 0020          MOVEA.L TFIL_OFF(A3),A0         get pointer to next     ( SPRxxx TM 6/14/82 )
1715                        *                                         byte in the buffer      ( SPRxxx TM 6/14/82 )
1716     000007F6 7601               MOVEQ   #1,D3                   set count = 1           ( SPRxxx TM 6/14/82 )
1717     000007F8 601E               BRA.S   H_BUFI                  let intr tfr finish     ( SPRxxx TM 6/14/82 )
1718                        *
1719     000007FA 4EB9 0000 H_NOBYTE JSR     DROPDMA                 FREE THE DMA CHANNEL
             0000
1720                        *                                         D4 has remaining
1721                        *                                         D3 has intended
1722     00000800 2744 0010          MOVE.L  D4,TCNT_OFF(A3)         put remaining into TCNT
1723     00000804 9684               SUB.L   D4,D3                   put # bytes tfr'd into D3
1724     00000806 D7AB 0020          ADD.L   D3,TFIL_OFF(A3)         update the fill pointer based on actual
1725                        *                                                 bytes tfr'd
1726     0000080A 4EB9 0000          JSR     STCLR                   MARK THE TFR DONE   AND LOG BRANCH
             0000
1727     00000810 6088               BRA     HISR_END                DONE!
1728
1729                        *
1730                        * INTERRUPT TRANSFER PROCESSING:
1731                        *
1732     00000812 4A2B 000D H_BUF    TST.B   TDIR_OFF(A3)            WHICH DIRECTION TRANSFER?
1733     00000816 6646               BNE.S   H_BUF0                  SKIP IF OUTPUT
1734
1735                        *
1736                        * BUFFERED INPUT:
1737                        *
1738     00000818 0800 000D H_BUFI   BTST    #13,D0                  IS BYTE IN SET?
1739     0000081C 6700 FF7C          BEQ     HISR_END                   IF NOT, DO NOTHING
1740     00000820 7800               MOVEQ   #0,D4                      ELSE GET THE BYTE
1741     00000822 1829 001F          MOVE.B  H_DATAIN(A1),D4
1742     00000826 0800 000B          BTST    #11,D0                  IS EOI SET WITH THIS BYTE?
1743     0000082A 6608               BNE.S   H_BUFI_0
1744     0000082C 082A 0000          BTST    #0,H_STAT3(A2)          is eor set
             0038
1745     00000832 6708               BEQ.S   H_BUFI_1                   IF NOT SKIP
1746     00000834 4A2B 000B H_BUFI_0 TST.B   TEND_OFF(A3)            SHOULD WE IGNORE EOI?
1747     00000838 6702               BEQ.S   H_BUFI_1                IF SO, SKIP
1748     0000083A 3404               MOVE.W  D4,D2                      ELSE USE TRICK TO MAKE TFR QUIT.
1749                        *                                                 - set term char to current char
1750     0000083C 10C4       H_BUFI_1 MOVE.B D4,(A0)+                SAVE CHARACTER IN BUFFER
1751     0000083E 2748 0020          MOVE.L  A0,TFIL_OFF(A3)         AND SAVE NEW FILL PTR
1752     00000842 5383               SUBQ.L  #1,D3                   HAVE WE TRANSFERED ALL CHARS?
1753     00000844 2743 0010          MOVE.L  D3,TCNT_OFF(A3)         SAVE TRANSFER COUNT...
1754     00000848 6F30               BLE.S   H_TI_TERM                  IF SO, SKIP
1755     0000084A B444               CMP.W   D4,D2                      ELSE TERMINATE ON CHARACTER?
1756     0000084C 6738               BEQ.S   H_TI_TERM               IF SO, SKIP
```

```
1757  0000084E 137C 0002           MOVE.B  #H_RHDF,H_AUXCMD(A1)  RELEASE HOLDOFF FOR NEXT BYTE
                0017
1758  00000854 08AA 0000           BCLR    #0,H_STAT3(A2)        clear eor flag
                0038
1759  0000085A 6000 FF3E H_BUF_NT  BRA     HISR_END              AND RETURN
1760
1760
1761                      *
1762                      * BUFFERED OUTPUT:
1763                      *
1764  0000085E 0800 000C H_BUF0    BTST    #12,D0                IS BYTE OUT SET ?
1765  00000862 6700 FF36           BEQ     HISR_END              IF NOT, DO NOTHING.
1766  00000866 5383                SUBQ.L  #1,D3                 IS THIS THE LAST CHARACTER?
1767  00000868 660C                BNE.S   H_BUF0_1              IF NOT, SKIP
1768  0000086A 4A2B 000B           TST.B   TEND_OFF(A3)          SHOULD WE SET EOI?
1769  0000086E 6706                BEQ.S   H_BUF0_1              IF NOT, SKIP
1770  00000870 137C 0008            MOVE.B #H_FEOI,H_AUXCMD(A1)  ELSE SET EOI WITH THE LAST BYTE
                0017
1771  00000876 1358 001F H_BUF0_1  MOVE.B  (A0)+,H_DATAOUT(A1)   SEND THE BYTE
1772  0000087A 2748 001C           MOVE.L  A0,TEMP_OFF(A3)       AND SAVE NEW EMPTY PTR
1773  0000087E 2743 0010           MOVE.L  D3,TCNT_OFF(A3)       SAVE TRANSFER COUNT...
1774  00000882 6ED6                BGT.S   H_BUF_NT              NO...DON'T TERMINATE
1775  00000884 6006                BRA.S   H_TO_TERM             YES...TERMINATE THE TFR
1776
1776
1777                      *
1778                      * INT AND FRW TRANSFER TERMINATION
1779                      *
1780  00000886 08EA 0007 H_TI_TERM BSET    #7,H_FLAGS(A2)        SET HOLDOFF INDICATOR
                0039
1781            0000 088C H_TO_TERM EQU     *
1782                      * tm     MOVE.B  #0,D0                 DISABLE OTHER INTERRUPTS
1783                      * tm     BSR     H_EIR
1784  0000088C 6100 FCD0           BSR     H_DISABLE             DISABLE BO/BI INTRPTS   ( hphp TM 1/19/83 )
1785  00000890 4EB9 0000           JSR     STCLR                 MARK THE BUFFER FINISHED
                0000
1786  00000896 6000 FF02           BRA     HISR_END              END OF ISR
```



```
1788                      *
1789                      * FAST R/W TRANSFER PROCESSING:
1790                      *
1791  0000089A 007C 2700 H_FRW     ORI     #$2700,SR             DISABLE ALL OTHER INTS
1792                      *                                     the pascal system will re-enable&RTE
1793  0000089E 4A2B 000D           TST.B   IDIR_OFF(A3)          WHICH DIRECTION TRANSFER?
1794  000008A2 667A                BNE.S   H_FRWO                SKIP IF OUTPUT
1795
1795
1795
1795
1796                      *
1797                      * FAST R/W INPUT:
1798                      *
1799  000008A4 7800       H_FRI     MOVEQ   #0,D4                 PRESET UPPER BYTE TO 0
1800  000008A6 E048                LSR     #8,D0                 REPOSITION REMAINING INT STAT BITS
1801  000008A8 0800 0005           BTST    #5,D0                 DO WE ALREADY HAVE BYTE IN?
1802  000008AC 661A                BNE.S   H_FRWI_2A             IF SO, SKIP
1803  000008AE 1029 0011 H_FRWI_1  MOVE.B  H_INTOSTAT(A1),D0     GET INTERRUPT STATUS
1804  000008B2 67FA                BEQ     H_FRWI_1              IF NOTHING, KEEP WAITING
1805  000008B4 0800 0003           BTST    #3,D0                 IS EOI SET?
1806  000008B8 660A                BNE.S   H_FRWI_2              IF SO, GO PROCESS BI AND END
1807  000008BA 0800 0005           BTST    #5,D0                 IS BYTE IN SET?
1808  000008BE 6608                BNE.S   H_FRWI_2A             IF SO, GO PROCESS IT
1809  000008C0 613C                BSR.S   H_FRW_OTHER           ELSE PROCESS OTHER INTERRUPTS
1810  000008C2 60EA                BRA     H_FRWI_1              AND KEEP WAITING
1811  000008C4 8029 0011 H_FRWI_2  OR.B    H_INTOSTAT(A1),D0     MAKE SURE WE GET THE BI BIT!
1812  000008C8 1829 001F H_FRWI_2A MOVE.B  H_DATAIN(A1),D4       GET THE DATA BYTE
1813  000008CC 10C4                MOVE.B  D4,(A0)+              SAVE IT IN THE BUFFER
1814  000008CE C07C 005F           AND     #$5F,D0               CLEAR BI STAT AND CHECK FOR OTHERS
1815  000008D2 6716                BEQ.S   H_FRWI_3              IF NO OTHER BITS SET, SKIP
1816  000008D4 6128                BSR.S   H_FRW_OTHER           ELSE PROCESS THE OTHERS
1817  000008D6 0800 000B           BTST    #3+8,D0               WAS EOI SET?
1818  000008DA 670E                BEQ.S   H_FRWI_3              IF NOT, SKIP
1819  000008DC 4A2B 000B           TST.B   TEND_OFF(A3)          SHOULD WE IGNORE EOI?
1820  000008E0 6708                BEQ.S   H_FRWI_3              IF SO, SKIP
1821  000008E2 3404                 MOVE.W D4,D2                 ELSE USE TRICK TO MAKE TFR TERMINATE
1822  000008E4 08EA 0000            BSET   #0,H_STAT3(A2)        SET EOR INDICATOR
                0038
1823  000008EA 5383       H_FRWI_3  SUBQ.L  #1,D3                 DONE?
1824  000008EC 6F00 001E           BLE     H_TBI_TRM             IF SO, GO QUIT
1825  000008F0 B444                CMP.W   D4,D2                 DOES CHAR MATCH TERMINATION CHAR?
1826  000008F2 6700 0018           BEQ     H_TBI_TRM             IF SO, GO QUIT
1827  000008F6 137C 0002           MOVE.B  #H_RHDF,H_AUXCMD(A1)  ELSE ASK FOR ANOTHER BYTE
                0017
1828  000008FC 60B0                BRA     H_FRWI_1              AND GO WAIT FOR IT
1829
1829
1830  000008FE 48E7 7000 H_FRW_OTHER MOVEM.L D1-D3,-(SP)           SAVE REGS FOR LATER
1831  00000902 4AC0                TAS     D0                    SET BIT 8 TO FORCE LOOK AT INTOSTAT
1832  00000904 615E                BSR.S   H_ISR0                PROCESS OTHER INTERRUPTS
1833  00000906 4CDF 000E           MOVEM.L (SP)+,D1-D3           RESTORE REGS FOR ANOTHER PASS
1834  0000090A 4E75                RTS
1835
1835
1836  0000090C 282B 0010 H_TBI_TRM MOVE.L  TCNT_OFF(A3),D4       get intended count
1837  00000910 2743 0010           MOVE.L  D3,TCNT_OFF(A3)       D3 has bytes not finished
```

```
1838    00000914 9883                SUB.L   D3,D4                  D4 has bytes transfered
1839    00000916 D9AB 0020           ADD.L   D4,TFIL_OFF(A3)        update fill pointer
1840    0000091A 6000 FF6A           BRA     H_TI_TERM              and finish
1841
1841
1841
1842                         *
1843                         * FAST R/W OUTPUT:
1844                         *
1845    0000091E 0800 000C H_FRW0    BTST    #12,D0                 DO WE ALREADY HAVE BYTE OUT?
1846    00000922 6614                BNE.S   H_FRW0_2               IF SO, GET STARTED
1847    00000924 1029 0011 H_FRW0_1  MOVE.B  H_INT0STAT(A1),D0      ELSE GET THE INTERRUPT STATUS
1848    00000928 67FA                BEQ     H_FRW0_1               IF NOTHING, KEEP WATCHING
1849    0000092A B03C 0090           CMP.B   #$90,D0                IS IT BO?
1850    0000092E 6708                BEQ.S   H_FRW0_2               IF SO, LETS GO!
1851    00000930 61CC                BSR     H_FRW_OTHER            ELSE GO PROCESS OTHER INTERRUPTS
1852    00000932 0800 000C           BTST    #12,D0                 IS BO LEFT?
1853    00000936 67EC                BEQ     H_FRW0_1               IF NOT, KEEP WATCHING
1854    00000938 B6BC 0000 H_FRW0_2  CMP.L   #1,D3                  IS THIS THE LAST BYTE?
                 0001
1855    0000093E 660C                BNE.S   H_FRW0_3               IF NOT, SKIP
1856    00000940 4A2B 000B           TST.B   TEND_OFF(A3)           SHOULD WE TAG WITH EOI?
1857    00000944 6706                BEQ.S   H_FRW0_3               IF NOT, SKIP
1858    00000946 137C 0008            MOVE.B #H_FEOT,H_AUXCMD(A1)   ELSE TAG IT!
                 0017
1859    0000094C 1358 001F H_FRW0_3  MOVE.B  (A0)+,H_DATAOUT(A1)    OUTPUT A BYTE FROM THE BUFFER
1860    00000950 5383                SUBQ.L  #1,D3                  DONE?
1861    00000952 6ED0                BGT     H_FRW0_1               IF NOT, WATCH FOR BO AGAIN
1862    00000954 262B 0010           MOVE.L  TCNT_OFF(A3),D3        \
1863    00000958 D7AB 001C           ADD.L   D3,TEMP_OFF(A3)        /  update empty pointer
1864    0000095C 42AB 0010           CLR.L   TCNT_OFF(A3)           clear the count
1865    00000960 6000 FF2A           BRA     H_TO_TERM              ELSE WE ARE DONE!
```



```
1867                         ********************************************************************************
1868                         *
1869                         *       ISR0
1870                         *       ISR1
1871                         *
1872                         *         THE FOLLOWING ROUTINE DOES ALL THE GRUNT WORK FOR THE ISR. IT IS
1873                         *         SEPARATED OUT SO IT CAN BE CALLED FROM BACKGROUND.
1874                         *
1875                         *         ENTRY:  D0 CONTAINS BOTH 9914 STATUS BYTES.
1876                         *
1877                         *         THE FOLLOWING CONDITIONS, IF THEY ARE THE CAUSE OF THE INTERRUPT, WILL
1878                         *         BE PROCESSED:
1879                         *
1880                         *                 END:  SET EOR LATCH.
1881                         *                 SPAS: CLEAR rsv INDICATOR, CHANGE PPOLL RESPONSE.
1882                         *                 RLC:  IF ENABLED, LOG BRANCH. (ENHANCEMENT)
1883                         *                 GET:  SET LATCH. IF ENABLED, LOG EOL BRANCH.  (ENHANCEMENT)
1884                         *                 VCG:  IF TCT, THEN REQUEST 9914 BECOME CONTROLLER.
1885                         *                                       IF ENABLED, LOG EOL BRANCH.
1886                         *                       IF PPC, THEN TELL 9914 TO PASS THROUGH NEXT SECONDARY.
1887                         *                       IF PPD OR SECONDARY, DO PP CONFIGURING.
1888                         *                 DCAS: SET DCAS LATCH.
1889                         *                       ELSE IF ENABLED, LOG EOL BRANCH.
1890                         *                 MA:   IF T/L IS ENABLED AND TADS/LADS IS TRUE, LOG EOL BRANCH.
1891                         *                 SRQ:  LOG EOL BRANCH, DISABLE SRQ INTERRUPT.
1892                         *                 IFC:  SET IFC LATCH. IF ENABLED, LOG EOL BRANCH. (ENHANCEMENT)
1893                         *                 NOTE: BO AND BI ARE NOT PROCESSED!
1894                         *
1895                         *         EXIT:   D0 HAS LEFT OVER BITS OF INTERRUPT STATUS. THE 'END' BIT
1896                         *                 WILL BE PROCESSED BY THIS ROUTINE BUT NOT CLEARED. ALL OTHER
1897                         *                 BITS PROCESSED WILL BE CLEARED.
1898                         *
1899                         *         USES:   D0-D3
1900                         *
1901                         *         HPL ROUTINE
1902                         *
1903                         ********************************************************************************
1904    00000964 E148        H_ISR0      LSL     #8,D0                  ALTERNATE ENTRY TO BUILD FULL STAT
1905    00000966 8029 0013               OR.B    H_INT1STAT(A1),D0      WORD IF ONLY BYTE 0 WAS READ.
1906    0000096A 806A 0034   H_ISR1      OR      H_INT0COPY(A2),D0      INCLUDE ANY SAVED BITS
1907    0000096E 3600                    MOVE.W  D0,D3                  SAVE COPY OF INT STATUS
1908    00000970 C66A 0036               AND     H_INTMSKSAV(A2),D3     KEEP ONLY THE ENABLED BITS IN D3
1909    00000974 322A 0036               MOVE.W  H_INTMSKSAV(A2),D1     TURN OFF THE ENABLED BITS IN D0
1910    00000978 4641                    NOT     D1
1911    0000097A C041                    AND     D1,D0
1912    0000097C 3540 0034               MOVE.W  D0,H_INT0COPY(A2)      SEND THESE BACK TO BACKGROUND
1913    00000980 142A 002C               MOVE.B  EIRB_OFF(A2),D2        GET CURRENT EIR BYTE
1914                         *
1915                         * PROCESS INTERRUPT CAUSES FROM INT1STAT:
1916                         *
1917    00000984 0883 000E               BCLR    #6+8,D3                ARE THERE ANY INT1 CAUSES?
1918    00000988 6700 0088               BEQ     H_NO_INT1              IF NOT, SKIP
1919
1920    0000098C 0883 0000               BCLR    #0,D3                  IFC INTERRUPT?
1921    00000990 670A                    BEQ.S   H_NO_IFC               IF NOT, SKIP
1922    00000992 08EA 0003                BSET   #3,H_FLAGS(A2)         ELSE SET IFC LATCH
                 0039
```

```
1923  00000998 6100 011A            BSR    H_LOG                  GO LOG INTERRUPT IF ENABLED
1924
1925  0000099C 0883 0001 H_NO_IFC   BCLR   #1,D3                   SRQ INTERRUPT?
1926  000009A0 6704                 BEQ.S  H_NO_SRQ                IF NOT, SKIP
1927  000009A2 6100 00F2            BSR    H_CHKSRQ               ELSE GO PROCESS SRQ INTERRUPT
1928
1929  000009A6 0883 0002 H_NO_SRQ   BCLR   #2,D3                   MA INTERRUPT?
1930  000009AA 670A                 BEQ.S  H_NO_MA
1931  000009AC 137C 0001            MOVE.B #H_DACR0,H_AUXCMD(A1) RELEASE DAC HOLDOFF
              0017
1932  000009B2 6100 00C0            BSR    H_CHKADDR              GO SEE IF WE SHOULD INTERRUPT
1933
1934  000009B6 0883 0003 H_NO_MA    BCLR   #3,D3                   DCL/SDC INTERRUPT?
1935  000009BA 6716                 BEQ.S  H_NO_DCL
1936  000009BC 137C 0001            MOVE.B #H_DACR0,H_AUXCMD(A1) RELEASE DAC HOLDOFF
              0017
1937  000009C2 08EA 0004            BSET   #4,H_FLAGS(A2)         SET DCL LATCH
              0039
1938  000009C8 0802 0001            BTST   #1,D2                   OTHER INTERRUPTS ENABLED?
1939  000009CC 6704                 BEQ.S  H_NO_DCL                IF NOT, SKIP
1940  000009CE 6100 00E4            BSR    H_LOG                   ELSE JUST LOG EOL BRANCH
1941
1942  000009D2 0883 0005 H_NO_DCL   BCLR   #5,D3                   UNIDENTIFIED COMMAND INTERRUPT?
1943  000009D6 6754                 BEQ.S  H_NO_UCG
1944  000009D8 1029 001D            MOVE.B H_CMDPASS(A1),D0       GET THE COMMAND
1945  000009DC 137C 0001            MOVE.B #H_DACR0,H_AUXCMD(A1) RELEASE DAC HOLDOFF
              0017
1946  000009E2 C07C 007F            AND    #$7F,D0                 IGNORE PARITY ON COMMANDS
1947  000009E6 B03C 0009            CMP.B  #TCT,D0                 IS THIS TAKE CONTROL?
1948  000009EA 6624                 BNE.S  H_NO_TCT                IF NOT, SKIP
1949                        *
1950                        *       waiting in ISR
1951                        *
1952                        *       trial for tct fix  11/29/81      4:47 PM
1953                        *
1954                        * tm        BTST  #0,H_FLAGS(A2)        IF THE 68000 IS TRYING TO PCT FROM
1955                        * tm        BNE.S H_TCTW1               ANOTHER SELECTCODE, THEN SKIP
1956  000009EC 3A3C 1FFF H_TCTW     MOVE.W #8191,D5                tttt JS 8/1/83
1957                        *
1958                        *     Count changed from 4095 to 8191 to allow for 16 MHz processors
1959                        *     This is really much more than enough since card only can respond
1960                        *     at 8 MHz rate.   tttt  JS 8/1/83
1961                        *
1962  000009F0 0829 0005 H_TCTWL     BTST  #5,H_ADRSSTAT(A1)       ELSE WAIT FOR ATN TO DROP
              0015
1963  000009F6 57CD FFF8            DBEQ  D5,H_TCTWL
1964  000009FA 137C 0011 H_TCTW1     MOVE.B #H_RQC,H_AUXCMD(A1)  REQUEST CONTROL FROM 9914
              0017
1965  00000A00 137C 000B            MOVE.B #H_GTS,H_AUXCMD(A1)  AND DROP ATN
              0017
1966  00000A06 0802 0006            BTST  #6,D2                    ENABLED TO INTERRUPT?
1967  00000A0A 6704                 BEQ.S H_NO_TCT                 IF NOT, SKIP
1968  00000A0C 6100 00A6            BSR   H_LOG                    ELSE LOG THE BRANCH
1969  00000A10 B03C 0005 H_NO_TCT   CMP.B  #PPC,D0                 PARALLEL POLL CONFIGURE?
1970  00000A14 6606                 BNE.S  H_NO_PPC                IF NOT, SKIP
1971  00000A16 137C 0014            MOVE.B #H_PTS,H_AUXCMD(A1)   ELSE PASS THRU NEXT SECONDARY
              0017
```



```
1972  00000A1C B03C 0015 H_NO_PPC   CMP.B   #PPU,D0                PARALLEL POLL UNCONFIGURE?
1973  00000A20 6706                 BEQ.S   H_PPE                  IF SO, TREAT SAME AS PPE
1974  00000A22 B03C 0060            CMP.B   #$60,D0                GENERAL SECONDARY?
1975  00000A26 6D04                 BLT.S   H_NO_PPE               IF NOT, THEN NOT PPE/PPD
1976  00000A28 6100 FA40 H_PPE      BSR     HPL_WTC3               ELSE GO SET PPOLL CONFIGURATION
1977            0000 0A2C H_NO_PPE   EQU     *                     PROCESS OTHER UCG VALUES HERE
1978
1979  00000A2C 0883 0007 H_NO_UCG   BCLR    #7,D3                  GET INTERRUPT?
1980  00000A30 6710                 BEQ.S   H_NO_GET               IF NOT, SKIP
1981  00000A32 137C 0001            MOVE.B #H_DACR0,H_AUXCMD(A1) ELSE RELEASE DAC HOLDOFF
              0017
1982  00000A38 08EA 0005            BSET    #5,H_FLAGS(A2)         AND SET GET LATCH
              0039
1983  00000A3E 6100 0074            BSR     H_LOG                  GO LOG INTERRUPT IF ENABLED
1984            0000 0A42 H_NO_GET   EQU     *                     PROCESS OTHER INT1 CAUSES HERE
1985                        *
1986                        * PROCESS INTERRUPT CAUSES FROM INTOSTAT:
1987                        *
1988  00000A42 0883 000F H_NO_INT1  BCLR    #7+8,D3                ARE THERE ANY INTO CAUSES?
1989  00000A46 6728                 BEQ.S   H_NO_INTO              IF NOT, SKIP
1990
1991  00000A48 0883 0009            BCLR    #1+8,D3                RLC INTERRUPT?
1992  00000A4C 6704                 BEQ.S   H_NO_RLC               IF NOT SKIP
1993  00000A4E 6100 0064            BSR     H_LOG                  GO LOG INTERRUPT IF ENABLED
1994
1995  00000A52 0883 000A H_NO_RLC   BCLR    #2+8,D3                SPAS INTERRUPT?
1996  00000A56 670C                 BEQ.S   H_NO_SPAS              IF NOT, SKIP
1997  00000A58 08AA 0006            BCLR    #6,H_FLAGS(A2)         ELSE INDICATE rsv IS NOW 0
              0039
1998  00000A5E 136A 003B            MOVE.B H_PPOLLMSK+1(A2),H_PPOLL(A1) AND UPDATE PPOLL
              001D
1999
2000  00000A64 0803 000B H_NO_SPAS  BTST    #3+8,D3                END INTERRUPT?
2001  00000A68 6706                 BEQ.S   H_NO_END               IF NOT, SKIP
2002  00000A6A 08EA 0000            BSET    #0,H_STAT3(A2)           ELSE SET EOR LATCH
              0038
2003
2004            0000 0A70 H_NO_END   EQU     *                     PROCESS OTHER INTO CAUSES HERE
2005  00000A70 3003                 H_NO_INTO MOVE.W  D3,D0        PUT REMAINING INT BITS BACK INTO D0
2006  00000A72 4E75                 RTS                            FOR CALLER.
```

```
2009                        ********************************************************************************
2010                        *
2011                        *       H_CHKADDR
2012                        *
2013                        *         SUBROUTINE TO CHECK FOR ADDRESS INTERRUPT
2014                        *
2015                        *         ENTRY:  D2.B = EIR BYTE
2016                        *
2017                        *         IF THE TLK (LST) BIT OF THE EIR BYTE IS TRUE AND THE 9914
2018                        *         IS ADDRESSED AS TALKER (LISTENER), THEN LOG AN EOL BRANCH.
2019                        *
2020                        *         HPL ROUTINE
2021                        *
2022                        ********************************************************************************
2023  00000A74 0802 0005 H_CHKADDR BTST   #5,D2                    INTERRUPT ON TALKER ENABLED (TLK)?
2024  00000A78 670A              BEQ.S  H_CHKA1                   IF NOT, SKIP
2025  00000A7A 0829 0001         BTST   #1,H_ADRSSTAT(A1)         ARE WE TALKER?
                  0015
2026  00000A80 6600 0032         BNE    H_LOG                     IF SO, GO LOG INTERRUPT
2027  00000A84 0802 0004 H_CHKA1 BTST   #4,D2                     INTERRUPT ON LISTEN ENABLED (LST)?
2028  00000A88 670A              BEQ.S  H_CHKA2                   IF NOT, SKIP
2029  00000A8A 0829 0002         BTST   #2,H_ADRSSTAT(A1)         ARE WE LISTENER?
                  0015
2030  00000A90 6600 0022         BNE    H_LOG                     IF SO, GO LOG INTERRUPT
2031  00000A94 4E75              H_CHKA2 RTS                      ELSE RETURN
2032                        ********************************************************************************
2033                        *
2034                        *       H_CHKSRQ
2035                        *
2036                        *         SUBROUTINE TO CHECK FOR SRQ INTERRUPT
2037                        *
2038                        *         ENTRY:  EIRB_OFF(A2) HAS ENABLE MASK
2039                        *
2040                        *         IF WE ARE CONTROLLER AND SRQ IS SET AND BIT 7 OF ENABLE BYTE
2041                        *         IS SET THEN LOG EITHER A NORMAL SRQ INTERRUPT.
2042                        *
2043                        *         HPL ROUTINE
2044                        *
2045                        ********************************************************************************
2046  00000A96 0829 0006 H_CHKSRQ BTST   #6,H_EXTSTAT(A1)         ARE WE CONTROLLER?
                  0005
2047  00000A9C 6614              BNE.S  H_CHKS2                   IF NOT, DO NOTHING
2048  00000A9E 0829 0002         BTST   #2,H_BUSSTAT(A1)          IS SRQ SET?
                  0017
2049  00000AA4 670C              BEQ.S  H_CHKS2                   IF NOT, DO NOTHING
2050  00000AA6 08AA 0007         BCLR   #7,EIRB_OFF(A2)           ARE WE ENABLED FOR SRQ?
                  002C
2051  00000AAC 6704              BEQ.S  H_CHKS2                   IF NOT, DO NOTHING
2052  00000AAE 6100 0004 H_CHKS1  BSR   H_LOG                     ELSE DO NORMAL LOGING
2053  00000AB2 4E75              H_CHKS2 RTS                      DONE CHECKING FOR SRQ
2054
2054
2054
2054
2055                        *
2056                        *       H_LOG    mark that an isr condition is
2057                        *                pending
```



```
2058                        *
2059  00000AB4 08EA 0001 H_LOG    BSET   #1,H_FLAGS(A2)           set condition
                  0039
2060  00000ABA 4E75              RTS
```

```
2063                            ********************************************************************************
2064                            *
2065                            *       H_TFR
2066                            *
2067                            *         DRIVER CALL FOR EXECUTION OF tfr STATEMENT
2068                            *
2069                            *         ENTRY:  CONDITIONS OTHER THAN NORMAL A1,A2 ARE:
2070                            *                 A3.L = POINTER TO TRANSFER INFORMATION
2071                            *
2072                            *         HPL ROUTINE ( MODIFIED BEYOND ALL RECOGNITION )
2073                            *
2074                            ********************************************************************************
2075  00000ABC 4EB9 0000 H_TFR  JSR     CHECK_TFR               wait for  tfr to finish ( timed )
                 0000
2076  00000AC2 137C 000B        MOVE.B #H_GTS,H_AUXCMD(A1)     MAKE SURE ATN IS FALSE
                 0017
2077  00000AC8 4A2B 000A        TST.B  T_BW_OFF(A3)            DON'T ALLOW WORD TRANSFERS
2078  00000ACC 6600 F778        BNE    H_NOWORD
2079  00000AD0 202B 0010        MOVE.L TCNT_OFF(A3),D0         GET COUNT
2080  00000AD4 4241             CLR.W  D1                      \
2081  00000AD6 122B 0009        MOVE.B TYSR_OFF(A3),D1          \    COMPUTE OFFSET INTO JUMP TABLE
2082  00000ADA D241             ADD.W  D1,D1                    )
2083  00000ADC 4EB9 0000        JSR    TESTDMA                  )   BASED ON TFR TYPE AND DMA PRESENCE
                 0000
2084  00000AE2 6704             BEQ.S  H_NODMA                 /
2085  00000AE4 0641 0014        ADDI.W #20,D1                  (
2086  00000AE8 41FA 0008 H_NODMA LEA   H_TBL,A0                \
2087  00000AEC D0F0 1000        ADDA.W 0(A0,D1),A0              INDEXED JUMP THRU TABLE
2088  00000AF0 4ED0             JMP    (A0)                    /
2089                            *
2090                            *         TRANSFER JUMP TABLE
2091                            *
2092                            *                                 --------------------  DMA is not installed or available
2093  00000AF2 F74C     H_TBL   DC.W   HTERR_B-H_TBL  serial  interrupt
2094  00000AF4 F750             DC.W   HTERR_D-H_TBL  serial  dma
2095  00000AF6 0148             DC.W   H_T_FHS-H_TBL  serial  fhs
2096  00000AF8 0148             DC.W   H_T_FHS-H_TBL  serial  fastest
2097  00000AFA F74C             DC.W   HTERR_B-H_TBL  serial  overlap
2098                            *                     --------------------
2099  00000AFC 0102             DC.W   H_T_INT-H_TBL  overlap interrupt
2100  00000AFE F750             DC.W   HTERR_D-H_TBL  overlap dma
2101  00000B00 010A             DC.W   H_T_BST-H_TBL  overlap fhs
2102  00000B02 010A             DC.W   H_T_BST-H_TBL  overlap fastest
2103  00000B04 0102             DC.W   H_T_INT-H_TBL  overlap overlap
2104                            *                     --------------------  DMA is installed
2105  00000B06 F74C             DC.W   HTERR_B-H_TBL  serial  interrupt
2106  00000B08 0028             DC.W   H_T_DMA-H_TBL  serial  dma
2107  00000B0A 0148             DC.W   H_T_FHS-H_TBL  serial  fhs
2108  00000B0C 0028             DC.W   H_T_DMA-H_TBL  serial  fastest
2109  00000B0E F74C             DC.W   HTERR_B-H_TBL  serial  overlap
2110                            *                     --------------------
2111  00000B10 0102             DC.W   H_T_INT-H_TBL  overlap interrupt
2112  00000B12 0028             DC.W   H_T_DMA-H_TBL  overlap dma
2113  00000B14 010A             DC.W   H_T_BST-H_TBL  overlap fhs
2114  00000B16 0028             DC.W   H_T_DMA-H_TBL  overlap fastest
2115  00000B18 0028             DC.W   H_T_DMA-H_TBL  overlap overlap
```



```
2117                            *
2118                            *         Transfer DMA
2119                            *
2120  00000B1A B0BC 0000 H_T_DMA  CMP.L  #1,D0                  \ USE INTR IF COUNT=1 ON DMA
                 0001
2121  00000B20 6700 00D2        BEQ    H_T_INT                 /
2122  00000B24 177C 0002        MOVE.B #TT_DMA,TACT_OFF(A3)    set tfr type to DMA
                 0007
2123  00000B2A 4A2B 000D        TST.B  TDIR_OFF(A3)            \ test for transfer direction
2124  00000B2E 6600 0098        BNE    H_TOD                   /
2125                            *
2126                            *         Transfer Input Dma:
2127                            *
2128            0000 0B32 H_TID   EQU    *
2129                            * tm     MOVEQ  #0,D0                   DISABLE CARD INTRPTS      hphp TM 1/25/83
2130                            * tm     BSR    H_EIR                                              hphp TM 1/25/83
2131  00000B32 137C 0093        MOVE.B #H_DAI1,H_AUXCMD(A1)    disable hpib card       ( SPR740 TM 5/24/82 )
                 0017
2132  00000B38 0829 0006        BTST   #6,H_EXTSTAT(A1)        ARE WE ACTIVE CONTROLLER?
                 0005
2133  00000B3E 6750             BEQ.S  H_TID_0                 IF SO, SKIP
2134                            *
2135                            *         non controller path
2136                            *
2137  00000B40 177C 0001        MOVE.B #TT_INT,TACT_OFF(A3)    fake tfr type as intr   ( SPR740 TM 5/24/82 )
                 0007
2138  00000B46 202B 0010        MOVE.L TCNT_OFF(A3),D0         copy count so this works( qqqq   TM 12/16/82 )
2139  00000B4A 4EB9 0000        JSR    STBSY                   set buf busy ( intr )   ( SPR740 TM 5/24/82 )
                 0000
2140  00000B50 6100 00C8        BSR    H_RHDF_S                OTHERWISE RELEASE HOLD OFF AND WAIT
2141  00000B54 6100 F7A6        BSR    H_WAIT_BI               AROUND FOR THE FIRST BYTE TO APPEAR.
2142  00000B58 08AA 0007        BCLR   #7,H_FLAGS(A2)          INSURE WON'T RE RELEASE HOLDOFF
                 0039
2143                            *
2144                            *         at this point BYTE IN is true.  IF EOI and EOI term     hphp   TM 1/26/83
2145                            *         are true then the tfr should be faked out as finished.  hphp   TM 1/26/83
2146                            *
2147  00000B5E 082A 0000        BTST   #0,H_STAT3(A2)          \ is EOI set                      hphp   TM 1/26/83
                 0038
2148  00000B64 672A             BEQ.S  H_TID_0                 /                                 hphp   TM 1/26/83
2149  00000B66 4A2B 000B        TST.B  TEND_OFF(A3)            \ is EOI term. enabled            hphp   TM 1/26/83
2150  00000B6A 6724             BEQ.S  H_TID_0                 /                                 hphp   TM 1/26/83
2151                            *
2152                            *         at this point - fake that the tfr is finished           hphp   TM 1/26/83
2153                            *
2154  00000B6C 4EB9 0000 H_TID_F  JSR     ITXFR                 get appropriate ptrs.             hphp   TM 1/26/83
                 0000
2155  00000B72 1829 001F        MOVE.B H_DATAIN(A1),D4         get data byte                     hphp   TM 1/26/83
2156  00000B76 10C4             MOVE.B D4,(A0)+                SAVE CHARACTER IN BUFFER          hphp   TM 1/26/83
2157  00000B78 2748 0020        MOVE.L A0,TFIL_OFF(A3)         AND SAVE NEW FILL PTR             hphp   TM 1/26/83
2158  00000B7C 5383             SUBQ.L #1,D3                   we have TFR'D ALL CHARS           hphp   TM 1/26/83
2159  00000B7E 2743 0010        MOVE.L D3,TCNT_OFF(A3)         SAVE TRANSFER COUNT...            hphp   TM 1/26/83
2160  00000B82 08EA 0007        BSET   #7,H_FLAGS(A2)          SET HOLDOFF INDICATOR             hphp   TM 1/26/83
                 0039
2161  00000B88 4EB9 0000        JSR     STCLR                 MARK THE BUFFER FINISHED          hphp   TM 1/26/83
                 0000
2162  00000B8E 6030             BRA.S  H_TID_E                 if done then finished             hphp   TM 1/26/83
```

```
2163                        *
2164                        *       common controller/non-controller path
2165                        *
2166            0000 0B90 H_TID_0   EQU   *                                  ( SPR740 TM 5/28/82 )
2167    00000B90 177C 0002          MOVE.B #TT_DMA,TACT_OFF(A3)  restore type as DMA  ( SPR740 TM 5/24/82 )
                 0007
2168    00000B96 202B 0010          MOVE.L TCNT_OFF(A3),D0       RESTORE D0
2169    00000B9A 5380               SUBQ.L #1,D0                 DMA CH SHOULD ONLY DO N-1 BYTES.
2170    00000B9C 4EB9 0000          JSR    GETDMA                TRY FOR DMA CHANNEL
                 0000
2171    00000BA2 3882               MOVE.W D2,(A4)               ARM THE CHANNEL
2172    00000BA4 6100 01F4          BSR    HD_STBSY              SET BUFFER BUSY, ETC    ( SPR740 TM 5/24/82 )
2173    00000BA8 4A2B 000B          TST.B  TEND_OFF(A3)          IF EOI TAG, THEN TELL   ( SPR740 TM 5/28/82 )
2174    00000BAC 6706               BEQ.S  H_TID_1               9914 TO HOLD OFF ON END ( SPR740 TM 5/28/82 )
2175    00000BAE 137C 0084          MOVE.B #H_HDFE1,H_AUXCMD(A1)                         ( SPR740 TM 5/28/82 )
                 0017
2176    00000BB4 137C 0003 H_TID_1  MOVE.B #H_HDFA0,H_AUXCMD(A1) TURN OFF HOLD OFF ON ALL
                 0017
2177    00000BBA 615E               BSR.S  H_RHDF_S              DO RHDF IF NECESSARY
2178    00000BBC 1343 0003          MOVE.B D3,3(A1)              ENABLE CARD FOR DMA
2179                        *
2180                        *       common exit for input DMA tfr
2181                        *
2182            0000 0BC0 H_TID_E   EQU   *                                  ( SPR740 TM 5/28/82)
2183    00000BC0 137C 0013          MOVE.B #H_DAI0,H_AUXCMD(A1)  enable hpib card     ( SPR740 TM 5/24/82 )
                 0017
2184    00000BC6 6018               BRA.S  H_DMA_W               DONE
2185                        *
2186                        *       Transfer Output Dma:
2187                        *
2188    00000BC8 4A2B 000B H_TOD    TST.B  TEND_OFF(A3)          IF EOI TAG IS SET, THEN LET DMA
2189    00000BCC 6702               BEQ.S  H_TOD_1                  DO ONLY N-1 BYTES AND DO THE LAST
2190    00000BCE 5380               SUBQ.L #1,D0                    UNDER INTERRUPT
2191    00000BD0 4EB9 0000 H_TOD_1  JSR    GETDMA                GET A DMA CHANNEL
                 0000
2192    00000BD6 3882               MOVE.W D2,(A4)               ARM THE CHANNEL
2193    00000BD8 6100 01C0          BSR    HD_STBSY              SET BUFFER BUSY, ETC
2194                        * tm    MOVEQ  #0,D0                 DISABLE USER INTERRUPTS   hphp TM 1/25/83
2195                        * tm    BSR    H_EIR                                           hphp TM 1/25/83
2196    00000BDC 1343 0003          MOVE.B D3,3(A1)              ENABLE CARD FOR DMA
2197                        *
2198                        *       H_DMA_W                      IF SERIAL THEN WAIT FOR  COMPLETION
2199                        *
2200    00000BE0 182B 0009 H_DMA_W  MOVE.B TUSR_OFF(A3),D4       \
2201    00000BE4 B83C 0005          CMP.B  #5,D4                  IS THE TRANSFER OVERLAP ?
2202    00000BE8 6C08               BGE.S  H_DMA_W2              /
2203    00000BEA 0C2B 00FF H_DMA_W1 CMPI.B #255,T_SC_OFF(A3)     IF NOT THEN WAIT TILL DONE
                 0005
2204    00000BF0 66F8               BNE.S  H_DMA_W1
2205    00000BF2 4E75       H_DMA_W2 RTS
```



```
2207                        *
2208                        *       Transfer INTERRUPT
2209                        *
2210    00000BF4 177C 0001 H_T_INT  MOVE.B #TT_INT,TACT_OFF(A3)  set tfr type to INTERRUPT
                 0007
2211    00000BFA 6006               BRA.S  H_T_BIC               go to common code
2212
2212
2212
2213                        *
2214                        *       Transfer BURST ( intr on 1st byte FHS on rest )
2215                        *
2216    00000BFC 177C 0003 H_T_BST  MOVE.B #TT_BURST,TACT_OFF(A3) set tfr type to BURST
                 0007
2217                        *       BRA.S  H_T_BIC               go to common code
2218
2218
2218
2219                        *
2220                        *       common interrupt and burst code
2221                        *
2222    00000C02 4EB9 0000 H_T_BIC  JSR    STBSY                 SET BUFFER BUSY, ETC
                 0000
2223    00000C08 4A2B 000D          TST.B  TDIR_OFF(A3)          \ test for transfer direction
2224    00000C0C 6622               BNE.S  H_TOI                 /
2225                        *
2226                        *       Transfer Input Interrupt or Transfer Input Burst
2227                        *
2228            0000 0C0E H_TII     EQU   *
2229                        * tm    MOVEQ  #9,D0                 ENABLE CARD FOR BYTE IN
2230                        * tm    BSR    H_EIR
2231    00000C0E 303C 2000          MOVE.W #$2000,D0             ENABLE CARD FOR BI  ( hphp TM 1/19/83 )
2232    00000C12 6100 F91E          BSR    H_ENABLE                                 ( hphp TM 1/19/83 )
2233    00000C16 6102               BSR.S  H_RHDF_S              should we release holdoff
2234    00000C18 60C6               BRA    H_DMA_W               see if tfr was serial - and wait
2235                        *                                    if it was
2236                        *
2237                        *
2238                        *
2239    00000C1A 08AA 0007 H_RHDF_S BCLR   #7,H_FLAGS(A2)        SHOULD WE RELEASE HOLDOFF?
                 0039
2240    00000C20 670C               BEQ.S  H_RHDF_S1             IF NOT, SKIP
2241    00000C22 08AA 0000          BCLR   #0,H_STAT3(A2)        clear eor flag
                 0038
2242    00000C28 137C 0002          MOVE.B #H_RHDF,H_AUXCMD(A1)  ELSE DO IT
                 0017
2243    00000C2E 4E75       H_RHDF_S1 RTS
2244                        *
2245                        *       Transfer Output Interrupt or Transfer Output Burst
2246                        *
2247            0000 0C30 H_TOI     EQU   *
2248                        * tm    MOVEQ  #4,D0                 ENABLE FOR BYTE OUT
2249                        * tm    BSR    H_EIR
2250    00000C30 303C 1000          MOVE.W #$1000,D0             ENABLE CARD FOR BO  ( hphp TM 1/19/83 )
2251    00000C34 6100 F8FC          BSR    H_ENABLE                                 ( hphp TM 1/19/83 )
2252    00000C38 60A6               BRA    H_DMA_W               wait if serial
```

```
2254                        ********************************************************************************
2255                        *                                Transfer FHS                                  *
2256                        ********************************************************************************
2257                        *
2258                        * WARNING: these FHS routines have been carefully optimized towards...
2259                        *   1. a close FHS coupling with Coyote (Greeley's new 913X Disc Controller)
2260                        *   2. efficient Series 200 to Series 200 transfers
2261                        *   3. efficient high-speed disc transfers
2262                        * While the inner loops can be tuned for higher-speed transfers with
2263                        * selected other devices, doing so will almost certainly compromise the
2264                        * above optimizations!  If you decide to optomize further, keep in mind
2265                        * that: 1) the internal and external HPIB's behave differently with the
2266                        * same FHS loop!!!, and 2) the 9914 & 9914A are programmed for different
2267                        * T1 delays!!!  Good Luck!
2268                        *                                                  -------
2269                        *
2270                        *
2271                        * special register assignments for the fast handshake transfer routines:
2272                        *
2273             0000 0005 fhs_eoi_bit  equ d5              always set to 0 for the eoi bit test
2274             0000 0006 fhs_BI_stat  equ d6              set to $20 for input  (int0stat w/ BI only)
2275             0000 0006 fhs_BO_stat  equ d6              set to $10 for output (int0stat w/ BO only)
2276             0000 0007 fhs_BI_bit   equ d7              set to 5 for input  (the BI bit number)
2277             0000 0007 fhs_BO_bit   equ d7              set to 4 for output (the BO bit number)
2278             0000 0004 fhs_int0stat equ a4              permanent pointer to the int0stat register
2279             0000 0005 fhs_auxcmd   equ a5              permanent pointer to the auxcmd   register
2280             0000 0006 fhs_datain   equ a6              permanent pointer to the datain   register
2281             0000 0006 fhs_dataout  equ fhs_datain      permanent pointer to the dataout  register
2282
2283
2284 00000C3A 177C 0004 H_T_FHS   MOVE.B #TT_FHS,TACT_OFF(A3)          set tfr type to FAST HANDSHAKE
             0007
2285 00000C40 4EB9 0000           JSR    STBSY                       make buffer busy
             0000
2286 00000C46 4EB9 0000           JSR    ITXFR                       set up pointers and registers
             0000
2287
2288 00000C4C 48E7 000E           movem.l fhs_int0stat-fhs_datain,-(sp)
2289 00000C50 7A00                moveq   #0,fhs_eoi_bit
2290 00000C52 49E9 0011           lea     h_int0stat(a1),fhs_int0stat
2291 00000C56 4BE9 0017           lea     h_auxcmd(a1),fhs_auxcmd
2292 00000C5A 4DE9 001F           lea     h_datain(a1),fhs_datain
2293
2294 00000C5E 1ABC 0093           move.b  #h_dai1,(fhs_auxcmd)        disable all card interrupts!
2295
2296 00000C62 4A2B 000D           TST.B   TDIR_OFF(A3)                which transfer direction?
2297 00000C66 6600 00B6           BNE     fto                         branch if output
2298
2299 00000C6A 61AE                BSR     H_RHDF_S                    input; release holdoff if necessary
2300
2301 00000C6C 7C20                moveq   #$20,fhs_BI_stat            int0stat with BI only
2302 00000C6E 7E05                moveq   #5,fhs_BI_bit               the BI bit number
2303
2304 00000C70 4A42                TST     D2                          termination character specified?
2305 00000C72 6A00 0070           BPL     fti                         branch if so
```



```
2307                        *
2308                        *         Transfer FHS in; NO termination character
2309                        *
2310 00000C76 1ABC 0003           move.b  #h_hdfa0,(fhs_auxcmd)       release holdoff on all
2311 00000C7A 5583                subq.l  #2,d3                       count-2!
2312 00000C7C 6C34                bge.s   fti_nt_i1                   initial BI test (n-1 bytes loop)
2313 00000C7E 601E                bra.s   fti_nt_i2                   initial BI test (last byte loop)
2314
2315                        *
2316                        * high-speed loop for n-1 bytes
2317                        *
2318 00000C80 1214        fti_nt_w1 move.b  (fhs_int0stat),d1          get card status
2319 00000C82 67FC                beq     fti_nt_w1                   loop until we get something
2320 00000C84 BC01                cmp.b   d1,fhs_BI_stat              BI status only?
2321 00000C86 6628                bne.s   fti_nt_s1                   if not, process other conditions
2322 00000C88 10D6        fti_nt_t1 move.b  (fhs_datain),(a0)+         transfer this data byte and request the next
2323 00000C8A 51CB FFF4           dbra    d3,fti_nt_w1                loop until lower count exhausted
2324
2325 00000C8E 4243                clr     d3                          clear lower count word only
2326 00000C90 5383                subq.l  #1,d3                       decrement the entire long count
2327 00000C92 6AEC                bpl     fti_nt_w1                   loop until entire long count exhausted
2328
2329                        *
2330                        * last byte handling
2331                        *
2332 00000C94 1214        fti_nt_w2 move.b  (fhs_int0stat),d1          get card status
2333 00000C96 67FC                beq     fti_nt_w2                   loop until we get something
2334 00000C98 BC01                cmp.b   d1,fhs_BI_stat              BI status only?
2335 00000C9A 6708                beq.s   fti_nt_t2                   if so, transfer the byte
2336 00000C9C 612C                bsr.s   j_fakeisr                   otherwise, process the other conditions
2337 00000C9E 0FAA 0034 fti_nt_i2 bclr    #fhs_BI_bit,h_int0copy(a2)  see if BI was logged
2338 00000CA2 67F0                beq     fti_nt_w2                   if not, keep waiting
2339 00000CA4 1ABC 0083 fti_nt_t2 move.b  #h_hdfa1,(fhs_auxcmd)       set holdoff on all again
2340 00000CA8 10D6                move.b  (fhs_datain),(a0)+          transfer the last data byte
2341 00000CAA 5283                addq.l  #1,d3                       correct remaining count
2342 00000CAC 6000 00C4           bra     h_tfi_trm                   go terminate
2343
2344                        *
2345                        * special status handling: n-1 bytes loop
2346                        *
2347 00000CB0 6118        fti_nt_s1 bsr.s   j_fakeisr                  process the other conditions
2348 00000CB2 0FAA 0034 fti_nt_i1 bclr    #fhs_BI_bit,h_int0copy(a2)  BI logged?
2349 00000CB6 67C8                beq     fti_nt_w1                   if not, keep testing status
2350 00000CB8 0B2A 0038           btst    #fhs_eoi_bit,h_stat3(a2)    BI logged; was EOI set?
2351 00000CBC 67CA                beq.s   fti_nt_t1                   if not, go transfer the byte
2352 00000CBE 4A2B 000B           tst.b   tend_off(a3)                BI w/ EOI; should we terminate?
2353 00000CC2 66E0                bne.s   fti_nt_t2                   if so, transfer the last byte
2354 00000CC4 0BAA 0038           bclr    #fhs_eoi_bit,h_stat3(a2)    otherwise, clear the eoi flag
2355 00000CC8 60BE                bra     fti_nt_t1                   and continue transferring bytes
2356
```

```
2358                        *
2359                        *         restoration of A5/6 required because fakeisr
2360                        *         can call a PASCAL user procedure
2361                        *
2362    00000CCA 4CDF 7000 j_fakeisr movem.l (sp)+,fhs_int0stat-fhs_datain restore the dedicated registers
2363    00000CCE 6100 F6BC          bsr     h_fakeisr
2364    00000CD2 43E7 000E          movem.l fhs_int0stat-fhs_datain,-(sp)
2365    00000CD6 49E9 0011          lea     h_int0stat(a1),fhs_int0stat
2366    00000CDA 4BE9 0017          lea     h_auxcmd(a1),fhs_auxcmd
2367    00000CDE 4DE9 001F          lea     h_datain(a1),fhs_datain
2368    00000CE2 4E75               rts
2369
2370
2371
2372                        *
2373                        *         Transfer FHS in; looking for a termination character
2374                        *
2375    00000CE4 7800       fti     moveq   #0,d4                       clear upper byte to enable word comparison
2376    00000CE6 6002               bra.s   fti_it                      make the initial BI test
2377
2378    00000CE8 61E0       fti_fi  bsr     j_fakeisr                   process the other conditions
2379    00000CEA 0FAA 0034 fti_it   bclr    fhs_BI_bit,h_int0copy(a2)   see if BI was logged
2380    00000CEE 6608               bne.s   fti_BI                      branch if so
2381
2382    00000CF0 1214       fti_w1  move.b  (fhs_int0stat),d1            get card status
2383    00000CF2 67FC               beq     fti_w1                      keep trying until we get something
2384    00000CF4 BC01               cmp.b   d1,fhs_BI_stat              BI status only?
2385    00000CF6 66F0               bne.s   fti_fi                      if not, process other conditions
2386
2387    00000CF8 1816       fti_BI  move.b  (fhs_datain),d4              get the data byte
2388    00000CFA 10C4               move.b  d4,(a0)+                    save it in the buffer
2389    00000CFC 0B2A 0038          btst    fhs_eoi_bit,h_stat3(a2)     was EOI set?
2390    00000D00 670E               beq.s   fti_tc                      branch if not
2391    00000D02 4A2B 000B          tst.b   tend_off(a3)                EOI was set; do we terminate?
2392    00000D06 6704               beq.s   fti_ceoi                    branch if not
2393    00000D08 3404                 move.w  d4,d2                       else use trick to make tfr terminate
2394    00000D0A 6004                 bra.s  fti_tc
2395    00000D0C 0BAA 0038 fti_ceoi bclr    fhs_eoi_bit,h_stat3(a2)     clear the eoi flag
2396    00000D10 5383       fti_tc  subq.l  #1,d3                       termination count expired?
2397    00000D12 6F5E               ble.s   h_tfi_trm                   branch if so
2398    00000D14 B444               cmp.w   d4,d2                       termination character match?
2399    00000D16 675A               beq.s   h_tfi_trm                   branch if so
2400    00000D18 1ABC 0002          move.b  #h_rhdf,(fhs_auxcmd)        else ask for another byte
2401    00000D1C 60D2               bra     fti_w1                      and go wait for it
2402
```



```
2404                        *
2405                        *         Transfer FHS out
2406                        *
2407    00000D1E 7C10       fto     moveq   #$10,fhs_BO_stat            int0stat with BO only
2408    00000D20 7E04               moveq   #4,fhs_BO_bit               the BO bit number
2409    00000D22 5583               subq.l  #2,d3                       count-2!
2410    00000D24 6C44               bge.s   fto_i1                      initial BO test (n-1 bytes loop)
2411    00000D26 601E               bra.s   fto_i2                      initial BO test (last byte loop)
2412
2413                        *
2414                        * high-speed loop for n-1 bytes
2415                        *
2416    00000D28 1214       fto_w1  move.b  (fhs_int0stat),d1            get card status
2417    00000D2A 67FC               beq     fto_w1                      loop until we get something
2418    00000D2C BC01               cmp.b   d1,fhs_BO_stat              BO status only?
2419    00000D2E 6636               bne.s   fto_s1                      if not, process other conditions
2420    00000D30 1C98       fto_t1  move.b  (a0)+,(fhs_dataout)          transfer a byte
2421    00000D32 51CB FFF4          dbra    d3,fto_w1                   loop until lower count exhausted
2422
2423    00000D36 4243               clr     d3                          clear lower count word only
2424    00000D38 5383               subq.l  #1,d3                       decrement the entire long count
2425    00000D3A 6AEC               bpl     fto_w1                      loop until entire long count exhausted
2426
2427                        *
2428                        * last byte handling
2429                        *
2430    00000D3C 1214       fto_w2  move.b  (fhs_int0stat),d1            get card status
2431    00000D3E 67FC               beq     fto_w2                      loop until we get something
2432    00000D40 BC01               cmp.b   d1,fhs_BO_stat              BO status only?
2433    00000D42 6708               beq.s   fto_t2                      if so, transfer the byte
2434    00000D44 6184               bsr     j_fakeisr                   otherwise, process the other conditions
2435    00000D46 0FAA 0034 fto_i2   bclr    fhs_BO_bit,h_int0copy(a2)   see if BO was logged
2436    00000D4A 67F0               beq     fto_w2                      if not, keep waiting
2437    00000D4C 4A2B 000B fto_t2   tst.b   tend_off(a3)                should we tag it with an EOI?
2438    00000D50 6704               beq.s   fto_ob                      branch if not
2439    00000D52 1ABC 0008          move.b  #h_teoi,(fhs_auxcmd)        else tag it!
2440    00000D56 1C98       fto_ob  move.b  (a0)+,(fhs_dataout)          output the last byte
2441
2442    00000D58 262B 0010          MOVE.L  TCNT_OFF(A3),D3             \
2443    00000D5C D7AB 001C          ADD.L   D3,TEMP_OFF(A3)             / update empty pointer
2444    00000D60 42AB 0010          CLR.L   TCNT_OFF(A3)                clear count
2445    00000D64 6020               BRA.S   H_TFO_TRM                   ELSE WE ARE DONE!
2446
2447                        *
2448                        * special status handling: n-1 bytes loop
2449                        *
2450    00000D66 6100 FF62 fto_s1   bsr     j_fakeisr                   process the other conditions
2451    00000D6A 0FAA 0034 fto_i1   bclr    fhs_BO_bit,h_int0copy(a2)   see if BO was logged
2452    00000D6E 66C0               bne     fto_t1                      if so, go transfer the byte
2453    00000D70 60B6               bra     fto_w1                      otherwise, keep waiting
2454
```

```
2456                            *
2457                            * FHS TRANSFER TERMINATION
2458                            *
2459    00000D72 282B 0010 H_TFI_TRM MOVE.L  TCNT_OFF(A3),D4          get intended count
2460    00000D76 2743 0010           MOVE.L  D3,TCNT_OFF(A3)          D3 has bytes not finished
2461    00000D7A 9883                SUB.L   D3,D4                    D4 has bytes transfered
2462    00000D7C D9AB 0020           ADD.L   D4,TFIL_OFF(A3)          update fill pointer
2463    00000D80 08EA 0007           BSET    #7,H_FLAGS(A2)           SET HOLDOFF INDICATOR
                 0039
2464    00000D86 022A 00CF H_TFO_TRM andi.b  #$cf,h_int0copy(a2)
                 0034
2465    00000D8C 1ABC 0013           move.b  #h_dai0,(fhs_auxcmd)     re-enable card interrupts!
2466    00000D90 4CDF 7000           movem.l (sp)+,fhs_int0stat-fhs_datain restore the dedicated registers
2467    00000D94 4EF9 0000           JMP     STCLR                    MARK BUFFER FINISHED & RETURN
                 0000
2468
2469
2470
2471                            ********************************************************************************
2472                            *
2473                            *       HD_STBSY
2474                            *
2475                            *         ROUTINE TO SET A DMA TFR BUFFER BUSY
2476                            *
2477                            *         ENTRY:
2478                            *                 D0.L = TRANSFER COUNT TO BE PUT IN TCNT_OFF(A2)
2479                            *                        AND TO BE ADDED TO E/F COUNT.
2480                            *                 A0.L = POINTER TO DMA TEMPS ( DMA1 OR DMA0 )
2481                            *                 A2.L = POINTER TO DRIVER TEMPS
2482                            *                 A3.L = POINTER TO BUFFER CTL BLOCK
2483                            *
2484                            *         HPL ROUTINE ( MODIFIED )
2485                            *
2486                            ********************************************************************************
2487    00000D9A 49FA F282 HD_STBSY  LEA     EXTH_EH_TDMA,A4         \
2488    00000D9E 4EF9 0000           JMP     DMA_STBSY               /  SAVE H_DMATERM ROUTINE IN DMA TEMPS
                 0000
```



```
2490                            END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ABORT_IO	ABS	205	00000015
CHECK_TFR	ABS	207	0000001B
CHECK_TIMER	ABS	209	00000021
DELAY_TIMER	ABS	208	0000001E
DMA_STBSY	ABS	203	00000010
DROPDMA	ABS	196	00000002
GETDMA	ABS	197	00000004
IODECLARATIONS	ABS	354	00000024
ITXFR	ABS	204	00000013
LOGEOT	ABS	200	0000000A
LOGINT	ABS	199	00000008
STBSY	ABS	201	0000000C
STCLR	ABS	202	0000000E
SYSGLOBALS	ABS	355	00000028
TESTDMA	ABS	198	00000006
WAIT_TFR	ABS	206	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
AVAIL_OFF	ABS	294			00000034
BADTMO	ABS	372			0000000B
BAD_RDS	ABS	380			00000013
BAD_SCT	ABS	381			00000014
BUF1_OFF	ABS	285			00000024
BUF0_OFF	ABS	286			00000028
BUF_BUSY	ABS	370			00000009
CCR	STREG	0			00000005
CRD_DWN	ABS	382			00000015
C_ADR	ABS	284			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	709			00000014
DFC	STREG	0			00000008
EIRB_OFF	ABS	287			0000002C
EOD_SEEN	ABS	383			00000016
ESC_CODE	ABS	391	SYSGLOBALS	+	FFFFFFFE

ESC_ERR	REL	869	00000254
EXTH_EH_CLR	REL	581	000000F4
EXTH_EH_END	REL	604	0000011A
EXTH_EH_INIT	REL	436	00000002
EXTH_EH_ISR	REL	445	00000010
EXTH_EH_PPOLL	REL	559	00000D0
EXTH_EH_RDB	REL	463	0000002C
EXTH_EH_RDS	REL	515	00000088
EXTH_EH_RDW	REL	485	00000050
EXTH_EH_SEND	REL	549	000000C0
EXTH_EH_SET	REL	571	000000E4
EXTH_EH_TDMA	REL	454	0000001E
EXTH_EH_TEST	REL	591	00000104
EXTH_EH_TFR	REL	539	000000B0
EXTH_EH_WTB	REL	475	00000040
EXTH_EH_WTC	REL	528	0000009E
EXTH_EH_WTW	REL	501	0000006E
EXTH_EXTH	REL	431	00000000
FHS_AUXCMD	AREG	2279	00000005
FHS_BI_BIT	DREG	2276	00000007
FHS_BI_STAT	DREG	2274	00000006
FHS_BO_BIT	DREG	2277	00000007
FHS_BO_STAT	DREG	2275	00000006
FHS_DATAIN	AREG	2280	00000006
FHS_DATAOUT	AREG	2281	00000006
FHS_EOI_BIT	DREG	2273	00000005
FHS_INTOSTAT	AREG	2278	00000004
FTI	REL	2375	00000CE4
FTI_BI	REL	2387	00000CF8
FTI_CEOI	REL	2395	00000D0C
FTI_FI	REL	2378	00000CE8
FTI_IT	REL	2379	00000CEA
FTI_NT_I1	REL	2348	00000CB2
FTI_NT_I2	REL	2337	00000C9E
FTI_NT_S1	REL	2347	00000CB0
FTI_NT_T1	REL	2322	00000C88
FTI_NT_T2	REL	2339	00000CA4
FTI_NT_W1	REL	2318	00000C80
FTI_NT_W2	REL	2332	00000C94
FTI_TC	REL	2396	00000D10
FTI_WL	REL	2382	00000CF0
FTO	REL	2407	00000D1E
FTO_I1	REL	2451	00000D6A
FTO_I2	REL	2435	00000D46
FTO_OB	REL	2440	00000D56
FTO_S1	REL	2450	00000D66
FTO_T1	REL	2420	00000D30
FTO_T2	REL	2437	00000D4C
FTO_W1	REL	2416	00000D28
FTO_W2	REL	2430	00000D3C
GET	ABS	706	00000008
GTL	ABS	703	00000001
HDMA_END	REL	1658	0000076E
HD_STBSY	REL	2487	00000D9A
HISR_END	REL	1686	0000079A
HPL_WTC	REL	1170	00000446
HPL_WTC0	REL	1172	0000044C

HPL_WTC1	REL	1178	00000452
HPL_WTC2	REL	1184	00000468
HPL_WTC3	REL	1186	0000046A
HPL_WTC4	REL	1227	000004AC
HPL_WTC5	REL	1233	000004BC
HR_CARD	REL	1085	000003DE
HR_CARD1	REL	1086	000003E0
HR_TEMP	REL	1078	000003D6
HTCTLOOP	REL	1613	00000708
HTERR_B	REL	856	0000023E
HTERR_D	REL	858	00000242
HWTCTBL	REL	1143	0000043A
H_ADDRESS	ABS	640	00000019
H_ADRSSTAT	ABS	637	00000015
H_ATN1A	REL	1451	0000060C
H_ATN1_0	REL	1458	00000622
H_ATN1_1	REL	1460	00000628
H_ATN1_2	REL	1462	00000632
H_ATN1_3	REL	1463	00000634
H_AUXCMD	ABS	639	00000017
H_BUF	REL	1732	00000812
H_BUFI	REL	1738	00000818
H_BUFI_0	REL	1746	00000834
H_BUFI_1	REL	1750	0000083C
H_BUFO	REL	1764	0000085E
H_BUFO_1	REL	1771	00000876
H_BUF_NT	REL	1759	0000085A
H_BUSSTAT	ABS	638	00000017
H_BYTTST	REL	1706	000007DC
H_CHKA1	REL	2027	00000A84
H_CHKA2	REL	2031	00000A94
H_CHKADDR	REL	2023	00000A74
H_CHKS1	REL	2052	00000AAE
H_CHKS2	REL	2053	00000AB2
H_CHKSRQ	REL	2046	00000A96
H_CLR	REL	1489	00000642
H_CLR_ATN	REL	1534	00000678
H_CMDPASS	ABS	642	0000001D
H_CRDREG	ABS	1050	00000000
H_C_TBL	REL	1497	00000656
H_DACR0	ABS	658	00000001
H_DACR1	ABS	659	00000081
H_DAI0	ABS	686	00000013
H_DAI1	ABS	687	00000093
H_DATAIN	ABS	644	0000001F
H_DATAOUT	ABS	645	0000001F
H_DISABLE	REL	1324	0000055E
H_DMATERM	REL	1630	00000712
H_DMATI_1	REL	1648	0000074A
H_DMATO	REL	1652	00000758
H_DMATO_1	REL	1657	00000768
H_DMA_W	REL	2200	00000BE0
H_DMA_W1	REL	2203	00000BEA
H_DMA_W2	REL	2205	00000BF2
H_DMYRTS	REL	1478	00000640
H_ED_COM	REL	1304	0000053C
H_ED_EXIT	REL	1311	00000556

H_EIR	REL	1254	000004C4
H_EIR2	REL	1267	000004EA
H_EIR3	REL	1273	00000502
H_EIR4	REL	1280	0000051C
H_EIR5	REL	1284	00000530
H_ENABLE	REL	1301	00000532
H_EOI	REL	1477	0000063A
H_EXTSTAT	ABS	632	00000005
H_FAKEISR	REL	1028	0000038C
H_FEOI	ABS	670	00000008
H_FGET0	ABS	666	00000006
H_FGET1	ABS	667	00000086
H_FLAGS	ABS	406	00000039
H_FRI	REL	1799	000008A4
H_FRW	REL	1791	0000089A
H_FRWI_1	REL	1803	000008AE
H_FRWI_2	REL	1811	000008C4
H_FRWI_2A	REL	1812	000008C8
H_FRWI_3	REL	1823	000008EA
H_FRWO	REL	1845	0000091E
H_FRWO_1	REL	1847	00000924
H_FRWO_2	REL	1854	00000938
H_FRWO_3	REL	1859	0000094C
H_FRW_OTHER	REL	1830	000008FE
H_GETSTAT	REL	1007	00000374
H_GTS	ABS	675	0000000B
H_G_STAT1	REL	1012	00000382
H_HDFA0	ABS	661	00000003
H_HDFA1	ABS	662	00000083
H_HDFE0	ABS	663	00000004
H_HDFE1	ABS	664	00000084
H_IFC	REL	1417	000005D2
H_IFC2	REL	1426	000005FA
H_INIT	REL	750	00000156
H_INIT0	REL	763	0000017E
H_INIT_C	REL	756	00000166
H_INIT_S	REL	776	0000018C
H_INT0COPY	ABS	400	00000034
H_INT0MASK	ABS	634	00000011
H_INT0STAT	ABS	633	00000011
H_INT1COPY	ABS	401	00000035
H_INT1MASK	ABS	636	00000013
H_INT1STAT	ABS	635	00000013
H_INTMSKSAV	ABS	402	00000036
H_INT_CA	ABS	621	00478000
H_ISR	REL	1674	00000770
H_ISR0	REL	1904	00000964
H_ISR1	REL	1906	0000096A
H_ISRDMA	REL	1694	000007AA
H_ISR_EX	REL	1689	000007A8
H_ISR_PM	ABS	283	0000001C
H_ISR_PR	ABS	281	00000014
H_ISR_SL	ABS	282	00000018
H_LOCAL	REL	1518	00000666
H_LOG	REL	2059	00000AB4
H_LON0	ABS	671	00000009
H_LON1	ABS	672	00000089

H_LSTERR	REL	862	0000024A
H_NBAF	ABS	665	00000005
H_NOBYTE	REL	1719	000007FA
H_NODMA	REL	2086	00000AE8
H_NOTACTL	REL	852	00000236
H_NOTSCTL	REL	854	0000023A
H_NOWORD	REL	860	00000246
H_NO_DCL	REL	1942	000009D2
H_NO_END	REL	2004	00000A70
H_NO_GET	REL	1984	00000A42
H_NO_IFC	REL	1925	0000099C
H_NO_INT0	REL	2005	00000A70
H_NO_INT1	REL	1988	00000A42
H_NO_MA	REL	1934	000009B6
H_NO_PPC	REL	1972	00000A1C
H_NO_PPE	REL	1977	00000A2C
H_NO_RLC	REL	1995	00000A52
H_NO_SPAS	REL	2000	00000A64
H_NO_SRQ	REL	1929	000009A6
H_NO_TCT	REL	1969	00000A10
H_NO_UCG	REL	1979	00000A2C
H_PPE	REL	1976	00000A28
H_PPOLL	ABS	643	0000001D
H_PPOLLMSK	ABS	417	0000003A
H_PTS	ABS	688	00000014
H_P_POLL	REL	1342	0000056E
H_R60UT	REL	1579	00000696
H_R60UT2	REL	1600	000006D8
H_R60UT3	REL	1604	000006E4
H_R60UT4	REL	1605	000006E8
H_R6TCT	REL	1608	000006F0
H_RDB	REL	811	000001E8
H_RDB0	REL	814	000001F6
H_RDB1	REL	818	0000020A
H_RDS	REL	1053	000003A6
H_RDSTBL	REL	1115	0000041A
H_RDS_CS	REL	1098	000003F0
H_RDS_ID	REL	1093	000003EA
H_RDS_LI	REL	1110	0000040E
H_RDS_ST	REL	1104	00000402
H_REN	REL	1395	000005C6
H_RHDF	ABS	660	00000002
H_RHDF_S	REL	2239	00000C1A
H_RHDF_S1	REL	2243	00000C2E
H_RLC	ABS	685	00000012
H_ROUTINE	ABS	1048	00000002
H_RPP0	ABS	678	0000000E
H_RPP1	ABS	679	0000008E
H_RQC	ABS	684	00000011
H_RQS	REL	1219	0000048C
H_RQS2	REL	1225	000004A4
H_RTL0	ABS	668	00000007
H_RTL1	ABS	669	00000087
H_RT_END	REL	1125	0000042C
H_RT_SIZ	ABS	1126	00000012
H_SCBSY	REL	848	0000022E
H_SC_ERR	REL	850	00000232

H_SET	REL	1365	000005A2
H_SET_ATN	REL	1449	00000602
H_SHDW0	ABS	691	00000016
H_SHDW1	ABS	692	00000096
H_SICO	ABS	680	0000000F
H_SIC1	ABS	681	0000008F
H_SPOLL	ABS	641	0000001B
H_SREO	ABS	682	00000010
H_SRE1	ABS	683	00000090
H_STAT3	ABS	403	00000038
H_STDL0	ABS	689	00000015
H_STDL1	ABS	690	00000095
H_SWRST0	ABS	656	00000000
H_SWRST1	ABS	657	00000080
H_S_TBL	REL	1374	000005B6
H_TBI_TRM	REL	1836	0000090C
H_TBL	REL	2093	00000AF2
H_TCA	ABS	676	0000000C
H_TCS	ABS	677	0000000D
H_TCTW	REL	1956	000009EC
H_TCTW1	REL	1964	000009FA
H_TCTWL	REL	1962	000009F0
H_TEMP	ABS	1049	00000001
H_TEST	REL	1559	0000068A
H_TEST_EX	REL	1583	00000694
H_TFI_TRM	REL	2459	00000D72
H_TFO_TRM	REL	2464	00000D86
H_TFR	REL	2075	00000ABC
H_TID	REL	2128	00000B32
H_TID_0	REL	2166	00000B90
H_TID_1	REL	2176	00000BB4
H_TID_E	REL	2182	00000BC0
H_TID_F	REL	2154	00000B6C
H_TII	REL	2228	00000C0E
H_TI_TERM	REL	1780	00000886
H_TLKERR	REL	864	0000024E
H_TMO	REL	866	00000252
H_TMO_ERR	REL	935	000002D0
H_TOD	REL	2188	00000BC8
H_TOD_1	REL	2191	00000BD0
H_TOI	REL	2247	00000C30
H_TONO	ABS	673	0000000A
H_TON1	ABS	674	0000008A
H_TO_TERM	REL	1781	0000088C
H_T_BIC	REL	2222	00000C02
H_T_BST	REL	2216	00000BFC
H_T_DMA	REL	2120	00000B1A
H_T_FHS	REL	2284	00000C3A
H_T_INT	REL	2210	00000BF4
H_VSTDL0	ABS	693	00000017
H_VSTDL1	ABS	694	00000097
H_WAIT_BI	REL	951	000002FC
H_WAIT_BO	REL	896	0000026A
H_WAIT_D1	REL	929	000002BC
H_WAIT_D2	REL	933	000002CE
H_WBIT	REL	983	0000034E
H_WBIT1	REL	985	00000354

```
H_WBIT2          REL      994                          0000036E
H_WBI_1          REL      956                          00000302
H_WBI_2          REL      961                          00000312
H_WBI_3          REL      972                          00000330
H_WBI_5          REL      979                          00000340
H_WBOT           REL      937                          000002D8
H_WBOT1          REL      939                          000002DE
H_WBOT2          REL      948                          000002F8
H_WBO_1          REL      901                          00000270
H_WBO_2          REL      906                          00000280
H_WBO_3          REL      917                          0000029E
H_WBO_5          REL      924                          000002AE
H_WTB            REL      834                          00000216
H_WTBO           REL      836                          0000021E
H_WTB1           REL      837                          00000224
H_WTC            REL     1137                          0000042C
H_WTC_PPC        REL     1192                          00000478
H_WTC_RST        REL     1174                          0000044E
H_WTC_SMA        REL     1177                          00000452
H_W_DONE         REL      928                          000002B8
INTLEV_1         REL      732                          00000144
IOE_ERROR        ABS      386                          FFFFFFE6
IOE_RSLT         ABS      388 IODECLARATIONS  +        FFFFFFBE
IOE_SC           ABS      389 IODECLARATIONS  +        FFFFFFBA
IO_MISC          ABS      384                          00000017
IO_SC            ABS      288                          0000002D
ISR_ENTRY        ABS      279                          00000000
J_FAKEISR        REL     2362                          00000CCA
LLO              ABS      708                          00000011
MA               ABS      293                          00000033
MA_W             ABS      292                          00000032
NOT_HPIB         ABS      363                          00000002
NOT_LSTN         ABS      377                          00000010
NOT_TALK         ABS      376                          0000000F
NO_ACTL          ABS      364                          00000003
NO_CARD          ABS      362                          00000001
NO_DATA          ABS      367                          00000006
NO_DMA           ABS      374                          0000000D
NO_DRV           ABS      373                          0000000C
NO_DVC           ABS      365                          00000004
NO_SCTL          ABS      379                          00000012
NO_SPACE         ABS      366                          00000005
NO_WORD          ABS      375                          0000000E
PPC              ABS      705                          00000005
PPD              ABS      716                          00000070
PPE              ABS      715                          00000060
PPU              ABS      710                          00000015
RCVR_BLK         ABS      392 SYSGLOBALS      +        FFFFFFF6
RDS_ERR          REL     1072                          000003D0
SC_BUSY          ABS      369                          00000008
SDC              ABS      704                          00000004
SET_INT_LEVEL    REL      726                          00000132
SFC              STREG      0                          00000009
SP               AREG       0                          00000007
SPD              ABS      712                          00000019
SPE              ABS      711                          00000018
SR               STREG      0                          00000006
```



```
SYSFLAG2         ABS      395                          FFFFFEDA
TACT_OFF         ABS      309                          00000007
TBSZ_OFF         ABS      332                          00000018
TBUF_OFF         ABS      331                          00000014
TCHR_OFF         ABS      328                          0000000E
TCNTERR          ABS      371                          0000000A
TCNT_OFF         ABS      330                          00000010
TCT              ABS      707                          00000009
TDIR_OFF         ABS      326                          0000000D
TEMP_OFF         ABS      333                          0000001C
TEND_OFF         ABS      324                          0000000B
TFIL_OFF         ABS      334                          00000020
TFR_ERR          ABS      368                          00000007
TIMEOUT          ABS      289                          0000002E
TIMER_PRESENT    ABS      394                          00000001
TMO_ERR          ABS      378                          00000011
TTMP_OFF         ABS      307                          00000000
TT_BURST         ABS      345                          00000003
TT_DMA           ABS      344                          00000002
TT_FHS           ABS      346                          00000004
TT_INT           ABS      343                          00000001
TUSR_OFF         ABS      310                          00000009
T_BU_OFF         ABS      322                          0000000A
T_DMAPRI         ABS      339                          00000030
T_PM_OFF         ABS      338                          0000002C
T_PR_OFF         ABS      335                          00000024
T_SC_OFF         ABS      308                          00000005
T_SL_OFF         ABS      337                          00000028
UNL              ABS      713                          0000003F
UNT              ABS      714                          0000005F
USER_ISR         ABS      280                          00000014
USP              STREG      0                          00000007
VBR              STREG      0                          0000000A
```

Description

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 6500 0086        bcs     done                    all done if no EXT table
   50
   51    00000018 3C1A     L1      move.w  (list)+,ext_offset      get relative address of an EXT symbol
   52    0000001A 56CA FFFC        dbne    ext_count,L1            scan till non-zero, meaning unresolved
   53    0000001E 677C             beq.s   done                    done if none
   54
   55    00000020 4A83             tst.l   def_count               length of DEF symbol table
   56    00000022 674E             beq.s   no_defs                 done if none
   57
   58    00000024 280C     get_def move.l  gvr_ptr,def_addr        save address of DEF record
```



```
   59    00000026 4281             clr.l   len                     find GVR of DEF record
   60    00000028 1214             move.b  (gvr_ptr),len           length of string
   61    0000002A 5441             addq.w  #2,len                  compute length of string part
   62    0000002C 0881 0000        bclr    #0,len
   63    00000030 D8C1             adda.w  len,gvr_ptr             skip over string
   64    00000032 9681             sub.l   len,def_count
   65    00000034 0814 0002        btst    #gvr_flag,(gvr_ptr)     check for exported symbol
   66    00000038 662C             bne.s   next_def                ignore if not present
   67
   68    0000003A 41F1 6000 compare lea    0(ext_base,ext_offset.w),ext_str  get addresses of two strings
   69    0000003E 2644             movea.l def_addr,def_str
   70
   71    00000040 7000             MOVEQ   #0,COND                 standard string compare          (rdq)
   72    00000042 1018             move.b  (ext_str)+,COND         get EXT symbol length            (rdq)
   73    00000044 3200             MOVE.W  COND,LEN                                                 (rdq)
   74    00000046 4841             SWAP    LEN                                                      (rdq)
   75    00000048 3200             MOVE.W  COND,LEN                                                 (rdq)
   76    0000004A 101B             MOVE.B  (DEF_STR)+,COND                                          (rdq)
   77    0000004C B200             cmp.b   COND,len                compare two lengths              (rdq)
   78    0000004E 6302             bls.s   str_comp
   79    00000050 1200             move.b  COND,len                get minimum of two string lengths(rdq)
   80    00000052 5301     str_comp subq.b #1,len                  loop if at least one character
   81    00000054 6508             bcs.s   cmp_end
   82    00000056 B10B     cmp_lp  cmpm.b  (def_str)+,(ext_str)+   compare string bodies
   83    00000058 56C9 FFFC        dbne    len,cmp_lp              loop till not equal or end of string
   84    0000005C 6606             bne.s   nomatch                 if string bodies are equal
   85    0000005E 4841     cmp_end SWAP    LEN                             then compare lengths     (rdq)
   86    00000060 B200             CMP.B   COND,LEN                                                 (rdq)
   87    00000062 6712             beq.s   match                   three way branch on comparison
   88    00000064 632A     nomatch bls.s   ext_low
   89
   90    00000066 4281     next_def clr.l  len                     DEF does not match an EXT
   91    00000068 122C 0001        move.b  offset(gvr_ptr),len     get length of GVR
   92    0000006C D8C1             adda.w  len,gvr_ptr
   93    0000006E 9681             sub.l   len,def_count           decrease size of DEF table to search
   94    00000070 62B2             bhi.s   get_def                 loop back if there is any left
   95
   96    00000072 4205     no_defs clr.b   match_resolve           resolved := false
   97    00000074 6026             bra.s   done                    no more DEF records
   98
   99    00000076 0045 0100 match  ori.w   #$0100,match_resolve    matched := true
  100    0000007A 2384 6000        move.l  def_addr,0(ext_base,ext_offset.w)   copy address of DEF to EXT
  101    0000007E 8F14             or.b    mask,(gvr_ptr)          mark DEF record as having been matched
  102    00000080 426A FFFE        clr.w   -2(list)                index := 0 to flag it as resolved
  103    00000084 6002             bra.s   nxtext2                 find next unresolved EXT
  104    00000086 3C1A     L2      move.w  (list)+,ext_offset      get relative address
  105    00000088 56CA FFFC nxtext2 dbne   ext_count,L2            scan till non zero or end of list
  106    0000008C 6608             bne.s   next_def                find next DEF record
  107    0000008E 600C             bra.s   done                    done if none
  108
  109    00000090 4205     ext_low clr.b   match_resolve           resolved := false
  110    00000092 6002             bra.s   nxtext3                 find next unresolved EXT
  111    00000094 3C1A     L3      move.w  (list)+,ext_offset      get relative address
  112    00000096 56CA FFFC nxtext3 dbne   ext_count,L3            scan till non zero or end of list
  113    0000009A 669E             bne.s   compare                 compare to same DEF record
  114
  115    0000009C 205F     done    movea.l (sp)+,return            return address
```

```
  116    0000009E 225F              movea.l (sp)+,a1              address of matched
  117    000000A0 245F              movea.l (sp)+,a2              address of resolved
  118    000000A2 1485              move.b  match_resolve,(a2)    return boolean results
  119    000000A4 E04D              lsr.w   #8,match_resolve
  120    000000A6 8B11              or.b    match_resolve,(a1)    matched := matched or match
  121    000000A8 4ED0              jmp     (return)
  122
  123                               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	85		0000005E
CMP_LP	REL	82		00000056
COMPARE	REL	68		0000003A
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
DFC	STREG	0		00000008
DONE	REL	115		0000009C
EXT_BASE	AREG	19		00000001
EXT_COUNT	DREG	29		00000002
EXT_LOW	REL	109		00000090
EXT_OFFSET	DREG	33		00000006
EXT_STR	AREG	17		00000000
GET_DEF	REL	58		00000024
GVR_FLAG	ABS	15		00000002
GVR_PTR	AREG	22		00000004
L1	REL	51		00000018
L2	REL	104		00000086
L3	REL	111		00000094
LEN	DREG	28		00000001
LIST	AREG	20		00000002
MASK	DREG	34		00000007
MATCH	REL	99		00000076
MATCHDEFEXT	REL	36		00000000
MATCH_RESOLVE	DREG	32		00000005
NEXT_DEF	REL	90		00000066
NOMATCH	REL	88		00000064
NO_DEFS	REL	96		00000072
NXTEXT2	REL	105		00000088

```
NXTEXT3          REL      112                    00000096
OFFSET           ABS       14                    00000001
RETURN           AREG      18                    00000000
SFC              STREG      0                    00000009
SP               AREG       0                    00000007
SR               STREG      0                    00000006
STR_COMP         REL       80                    00000052
USP              STREG      0                    00000007
VBR              STREG      0                    0000000A
```

MATCHSTR

Description

MATCHSTR contains low-level assembly language string functions.

```
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
```



```
   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 67C4               beq.s   scanx
  113    00000080 51CA FFF0          dbra    d2,scan1
  114    00000084 4E75       scanx   rts
  115             0000 0086 scanloop         equ     *
```

```
000000E6 9083              sub.l   d3,d0
000000E8 5280              addq.l  #1,d0
000000EA 6D00 0036         blt     sfexit  pos in range ?

000000EE 9041              sub     d1,d0   is str2 longer than
000000F0 6D00 0030         blt     sfexit  remaining str1 ?

000000F4 4A41              tst.w   d1
000000F6 6726              beq.s   ssexit2 str2 is null so match

000000F8 2448              movea.l a0,a2   save str1 ptr
000000FA D0C3              adda.w  d3,a0   start source compare

000000FC 1C19              move.b  (a1)+,d6        first character
000000FE 5541              subq    #2,d1

00000100 BC18      scl1    cmp.b   (a0)+,d6
00000102 57C8 FFFC scl2    dbeq    d0,scl1
00000106 661A              bne.s   sfexit  found it ?

00000108 2648              movea.l a0,a3   temp str1
0000010A 2849              movea.l a1,a4   temp str2

0000010C 3A01              move.w  d1,d5   remaining str2 bytes
0000010E 6D08              blt.s   ssexit  str2 is 1 char

00000110 B90B      scl3    cmpm.b  (a3)+,(a4)+
00000112 56CD FFFC         dbne    d5,scl3
00000116 66EA              bne     scl2
00000118 260B      ssexit  move.l  a3,d3
0000011A 968A              sub.l   a2,d3
0000011C 4E75              rts
0000011E 2603      ssexit2 move.l  d3,d3   set condition code
00000120 4E75              rts

00000122 7600      sfexit  moveq   #0,d3   cursor to zero
00000124 4E75              rts
                   *
                           def     matchstr_beforestr
00000126 0000              dc.w    0
          0000 00C8 matchstr_beforestr equ     *
00000128 4E56 0000         link    a6,#0
0000012C 42AE 001A         clr.l   afunc(a6)       func:=0
00000130 206E 0014         movea.l as1(a6),a0      a0:=^s1
00000134 7000              moveq   #0,d0
00000136 1010              move.b  (a0),d0         d0:=strlen(s1)
00000138 242E 000C         move.l  an(a6),d2    if n>=0
0000013C 6D2A              blt.s   before1      then
                   *
                   * count
                   *
0000013E 262E 0010         move.l  ac(a6),d3       d3:=c
00000142 6F00 FF6A         ble     aret    check cursor<=0
00000146 226E 0008         movea.l as2(a6),a1      a1:=^s1
0000014A 7200              moveq   #0,d1
0000014C 1219              move.b  (a1)+,d1        d1:=strlen(s2)
0000014E 6710              beq.s   before0
```



```
00000150 6100 FF7E         bsr     scan
00000154 6702              beq.s   bgood
00000156 9681              sub.l   d1,d3   move to front of match
00000158 2D43 001A bgood   move.l  d3,afunc(a6)
0000015C 6000 FF50         bra     aret
                   *
                   * count but no string
                   *
00000160 9682      before0 sub.l   d2,d3           c:=c-n
00000162 6F00 FF4A         ble     aret    c :: 0
00000166 60F0              bra     bgood
                   *                                       else
                   * no count given
                   *
00000168 226E 0008 before1 movea.l as2(a6),a1
0000016C 7200              moveq   #0,d1
0000016E 1219              move.b  (a1)+,d1
00000170 660A              bne.s   before2
                   *
                   * no count no string
                   *
00000172 7001              moveq   #1,d0           func:=1
00000174 2D40 001A         move.l  d0,afunc(a6)
00000178 6000 FF34         bra     aret
                   *
                   * no count with string
                   *
0000017C 7401      before2 moveq   #1,d2   count 1
0000017E 262E 0010         move.l  ac(a6),d3
00000182 6F00 FF2A         ble     aret    cursor out of range?
00000186 6100 FF48         bsr     scan
0000018A 6700 FF22         beq     aret    must match at least once
0000018E 2D43 001A before3 move.l  d3,afunc(a6)
00000192 93AE 001A         sub.l   d1,afunc(a6)    move to front of match
00000196 7401              moveq   #1,d2   reset count
00000198 6100 FF36         bsr     scan
0000019C 66F0              bne     before3
0000019E 6000 FF0E         bra     aret

          0000 001E mf       equ     30
          0000 001C ms1m     equ     28
          0000 0018 ms1      equ     24
          0000 0014 mc       equ     20
          0000 0010 mk       equ     16
          0000 000C ms2      equ     12
          0000 0008 ms3      equ     8
          0000 0016 mr       equ     22
                           def     matchstr_changestr
000001A2 0000              dc.w    0
          0000 0144 matchstr_changestr equ     *
000001A4 4E56 0000         link    a6,#0

000001A8 42AE 001E         clr.l   mf(a6)  function result 0

000001AC 282E 0014         move.l  mc(a6),d4       cursor
000001B0 6F00 0166         ble     chgret
```

```
PAGE   5 [3.0] 7/26/84 12:08:06  ASSEMBLY OF MATCHSTR.TEXT                                   *** File name: MATCHSTR ***

  230    00000154 206E 0018          movea.l ms1(a6),a0
  231    00000158 7000               moveq   #0,d0
  232    0000015A 1010               move.b  (a0),d0
  233    0000015C 9084               sub.l   d4,d0
  234    0000015E 5280               addq.l  #1,d0
  235    00000160 6D00 0156          blt     chgret  cursor in range ?
  236
  237    00000164 2A2E 0010          move.l  mk(a6),d5       counter
  238    00000168 6700 0046          beq     chgzcnt
  239    0000016C 6F00 0054          ble     chgncnt
  240
  241                        *        have count value
  242
  243    00000170 226E 000C          movea.l ms2(a6),a1
  244    00000174 4A11               tst.b   (a1)
  245    00000176 6700 0082          beq     chgcnil1
  246
  247                        *        have count and s2 and maybe s3
  248                        *        replace the next n occurences of s2 with s3
  249
  250    0000017A 6014               bra.s   chgl2
  251    0000017C 48E7 0400 chgl1    movem.l d5,-(sp)        save count
  252    00000180 6100 00AE          bsr     chgflds
  253    00000184 4CDF 0020          movem.l (sp)+,d5        get count
  254    00000188 2D44 001E          move.l  d4,mf(a6)       set func
  255    0000018C 6700 012A          beq     chgret
  256    00000190 51CD FFEA chgl2    dbra    d5,chgl1
  257    00000194 6000 0122          bra     chgret
  258
  259                        *        count but no s2
  260                        *        replace next count chars with s3
  261    00000198 7000               moveq   #0,d0
  262    0000019A 1010               move.b  (a0),d0
  263    0000019C 2600               move.l  d0,d3   final length of s1
  264    0000019E 9685               sub.l   d5,d3
  265    000001A0 246E 0008          movea.l ms3(a6),a2
  266    000001A4 7400               moveq   #0,d2
  267    000001A6 1412               move.b  (a2),d2
  268    000001A8 D682               add.l   d2,d3
  269    000001AA 6D00 010C          blt     chgret  count is too big
  270
  271                        *        count is zero
  272                        *        insert s3 at cursor
  273
  274    000001AE 0000       nilstr  dc.w    0
  275                        *                make ms2 a dummy nilstring
  276    000001B0 2D7C 0000 chgzcnt  move.l  #nilstr,ms2(a6)
                 01AE 000C
  277    000001B8 6176               bsr.s   chgflds
  278    000001BA 2D44 001E          move.l  d4,mf(a6)
  279    000001BE 6000 00F8          bra     chgret
  280                        *
  281                        *        no count
  282                        *
  283    000001C2 226E 000C chgncnt  movea.l ms2(a6),a1
  284    000001C6 4A11               tst.b   (a1)
  285    000001C8 6716               beq.s   chgnil1
```

```
PAGE   6 [3.0] 7/26/84 12:08:06  ASSEMBLY OF MATCHSTR.TEXT                                   *** File name: MATCHSTR ***

  286
  287                        *        no count but has s2 might have s3
  288                        *        replace all occurences of s2 with s3
  289
  290    000001CA 6164               bsr.s   chgflds
  291    000001CC 2D44 001E          move.l  d4,mf(a6)
  292    000001D0 6700 00E6          beq     chgret  must change at least one
  293    000001D4 615A       chgncnt1 bsr.s  chgflds
  294    000001D6 2D44 001E          move.l  d4,mf(a6)       set func value
  295    000001DA 6700 00DC          beq     chgret
  296    000001DE 60F4               bra     chgncnt1
  297
  298                        *        no count no s2
  299
  300    000001E0 246E 0008 chgnil1  movea.l ms3(a6),a2
  301    000001E4 4A12               tst.b   (a2)
  302    000001E6 6706               beq.s   chgnil2
  303
  304                        *        no count only s3
  305                        *        replace rest of s1 with s3
  306
  307    000001E8 1084               move.b  d4,(a0) chop s1 to cursor
  308    000001EA 5310               subq.b  #1,(a0)
  309    000001EC 60C2               bra     chgzcnt add s3
  310                        *
  311                        * no count no strings
  312                        *        delete remainder of s1
  313
  314    000001EE 1084       chgnil2 move.b  d4,(a0) set s1 length
  315    000001F0 5310               subq.b  #1,(a0)
  316    000001F2 2D44 001E          move.l  d4,mf(a6) set func value
  317    000001F6 6000 00C0          bra     chgret
  318
  319                        *        have count no s2
  320                        *        replace count bytes with s3
  321
  322    000001FA 9085       chgcnil1 sub.l   d5,d0   d0 is #bytes after delete
  323    000001FC 6D00 00BA          blt     chgret
  324
  325    00000200 246E 0008          movea.l ms3(a6),a2 addr and size of s3
  326    00000204 7400               moveq   #0,d2
  327    00000206 141A               move.b  (a2)+,d2
  328
  329    00000208 7600               moveq   #0,d3
  330    0000020A 1610               move.b  (a0),d3 will it fit
  331    0000020C 9645               sub     d5,d3
  332    0000020E D642               add     d2,d3   final size of s1
  333    00000210 B62E 001C          cmp.b   ms1m(a6),d3
  334    00000214 6200 00A2          bhi     chgret
  335
  336    00000218 2E05               move.l  d5,d7   apparent size of s2
  337
  338    0000021A D0C4               adda    d4,a0   cursor addr
  339    0000021C 47F0 5000          lea     0(a0,d5.w),a3   after delete
  340    00000220 6100 0064          bsr     chgfi   do it
  341    00000224 2D44 001E          move.l  d4,mf(a6)
  342    00000228 6000 008E          bra     chgret
```

```
  343
  344    0000022C 7800       chgbad  moveq   #0,d4   cursor to zero
  345    0000022E 4E75               rts
  346                        *
  347                        *       do one change
  348                        *
  349    00000230 206E 0018 chgflds movea.l ms1(a6),a0     source string
  350    00000234 7000               moveq   #0,d0
  351    00000236 1010               move.b  (a0),d0       s length
  352    00000238 2600               move.l  d0,d3   final length of s
  353    0000023A 9084               sub.l   d4,d0
  354    0000023C 5280               addq.l  #1,d0
  355    0000023E 6DEC               blt     chgbad  pos in range ?
  356
  357    00000240 226E 000C          movea.l ms2(a6),a1      old string
  358    00000244 7200               moveq   #0,d1
  359    00000246 1219               move.b  (a1)+,d1        old lenght
  360    00000248 2E01               move.l  d1,d7   save it for later
  361
  362    0000024A 246E 0008          movea.l ms3(a6),a2      new string
  363    0000024E 7400               moveq   #0,d2
  364    00000250 141A               move.b  (a2)+,d2        new length
  365
  366    00000252 9641               sub     d1,d3
  367    00000254 D642               add     d2,d3
  368    00000256 B62E 001C          cmp.b   msim(a6),d3     will it all fit ?
  369    0000025A 62D0               bhi     chgbad
  370
  371    0000025C D0C4               adda.w  d4,a0   start source compare
  372
  373    0000025E 4A01               tst.b   d1
  374    00000260 6700 006E          beq     chgins          0 length so match
  375
  376    00000264 9041               sub     d1,d0   is old longer than
  377    00000266 6DC4               blt     chgbad  remaining source ?
  378
  379    00000268 1C19               move.b  (a1)+,d6        first character
  380    0000026A 5541               subq    #2,d1
  381
  382    0000026C BC18       chg1    cmp.b   (a0)+,d6
  383    0000026E 57C8 FFFC  chg2    dbeq    d0,chg1
  384    00000272 6644               bne.s   chgret  found it ?
  385
  386    00000274 2648               movea.l a0,a3   temp source
  387    00000276 2849               movea.l a1,a4   temp old
  388
  389    00000278 3A01               move.w  d1,d5   remaining old bytes
  390    0000027A 6D08               blt.s   chgf0   old is 1 char
  391
  392    0000027C B90B       chg3    cmpm.b  (a3)+,(a4)+
  393    0000027E 56CD FFFC          dbne    d5,chg3
  394    00000282 66EA               bne     chg2
  395
  396    00000284 5388       chgf0   subq.l  #1,a0
  397
  398    00000286 286E 0018 chgf1    movea.l ms1(a6),a4       string s
  399    0000028A 1883               move.b  d3,(a4)
```



```
  400    0000028C 4A40               tst.w   d0
  401    0000028E 6700 001C          beq     chgcpy1
  402    00000292 9E42               sub.w   d2,d7
  403    00000294 6716               beq.s   chgcpy1
  404    00000296 6E2A               bgt.s   chgsml
  405                        * new string is greater than old
  406    00000298 49F4 3001          lea     1(a4,d3.w),a4     end s + 1
  407    0000029C 47F4 7000          lea     0(a4,d7.w),a3   source
  408    000002A0 5340               subq.w  #1,d0   count
  409
  410    000002A2 1923       chgins1 move.b  -(a3),-(a4)
  411    000002A4 51C8 FFFC          dbra    d0,chgins1
  412    000002A8 6002               bra.s   chgcpy1
  413
  414    000002AA 10DA       chgcpy  move.b  (a2)+,(a0)+
  415    000002AC 51CA FFFC  chgcpy1 dbra    d2,chgcpy
  416
  417    000002B0 91EE 0018          suba.l  ms1(a6),a0
  418    000002B4 2808               move.l  a0,d4           new cursor value
  419    000002B6 4E75               rts
  420
  421    000002B8 4E5E       chgret  unlk    a6
  422    000002BA 205F               movea.l (sp)+,a0
  423    000002BC DEFC 0016          adda.w  #mr,sp
  424    000002C0 4ED0               jmp     (a0)
  425                        *
  426                        * new string is smaller than old
  427              0000 02C2 chgsml  equ     *
  428    000002C2 284B               movea.l a3,a4
  429    000002C4 98C7               suba.w  d7,a4
  430    000002C6 5340               subq.w  #1,d0
  431
  432    000002C8 18DB       chgdel1 move.b  (a3)+,(a4)+
  433    000002CA 51C8 FFFC          dbra    d0,chgdel1
  434    000002CE 60DC               bra     chgcpy1
  435
  436              0000 02D0 chgins  equ     *
  437    000002D0 2648               movea.l a0,a3
  438    000002D2 60B2               bra     chgf1
  439
  440              0000 0014 bf      equ     20
  441              0000 0010 bs1     equ     16
  442              0000 000C bc      equ     12
  443              0000 0008 bs2     equ     8
  444              0000 000C bargs   equ     12
  445
  446                        def     matchstr_breakstr
  447    000002D4 0000               dc.w    0
  448              0000 02D6 matchstr_breakstr  equ     *
  449    000002D6 4E56 0000          link    a6,#0
  450    000002DA 42AE 0014          clr.l   bf(a6)          set func to 0
  451
  452    000002DE 282E 000C          move.l  bc(a6),d4       cursor pos
  453    000002E2 6F3A               ble.s   bsret
  454
  455    000002E4 206E 0010          movea.l bs1(a6),a0
  456    000002E8 2848               movea.l a0,a4   save addr of s1
```

```
PAGE   9 [3.0] 7/26/84 12:08:06  ASSEMBLY OF MATCHSTR.TEXT                    *** File name: MATCHSTR ***

  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 232E 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 2348              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 [3.0] 7/26/84 12:08:06  ASSEMBLY OF MATCHSTR.TEXT                    *** File name: MATCHSTR ***

  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
```

Description

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

```
PAGE   1 [3.0] 12/26/84 21:00:53  ASSEMBLY OF MODIV.TEXT                                  *** File name: MODIV ***

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
```

```
PAGE   2 [3.0] 12/26/84 21:00:53  ASSEMBLY OF MODIV.TEXT                                  *** File name: MODIV ***

   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 6AA0                          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
DFC	STREG	0		00000008
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
SFC	STREG	0		00000009
SP	AREG	0		00000007
SR	STREG	0		00000006
USP	STREG	0		00000007
VALRANGE	REL	16		00000002
VBR	STREG	0		00000000A
ZERODIV	REL	42		0000003A

NEWWORDS

Description

NEWWORDS provides the procedure which implements NEW (if $HEAPDISPOSE OFF$) called NEWBYTES, the UCSD 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*, not *words* (a *word* is 2 bytes or 16 bits).

```
PAGE   1 [3.0] 12/26/84 21:00:10  ASSEMBLY OF NEWWORDS.TEXT                                  *** File name: NEWWORDS ***

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
```

Description

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.

```
PAGE   1 [3.0] 12/26/84 20:59:24  ASSEMBLY OF POWERUP.TEXT                                  *** File name: POWERUP ***

PASS 1 COMPLETE. ERRORS: 0
    1                           *
    2                           *       INTERRUPT, TRAP, AND EXCEPTION HANDLER
    3                           *
    4                           *    (rdq) changes for Rev 3.0
    5                           *
    6                           *     CHANGES FOR PARITY SUPPORT  -- JS 9/12/83
    7                           *
    8                                   DEF     ASM_SETINTLEVEL,ASM_INTLEVEL
    9                                   DEF     ASM_INITVECTS,RESETX,ASM_CLOSEFILES
   10                                   DEF     EXCP_PC,EXCP_LINE
   11                                   DEF     GRAPHICSBASE,GRAPHICSFLAG,ALPHAFLAG,FLTPTHDW
   12                                   DEF     ASM_IAND,ASM_IOR
   13                                   DEF     ERR_INFO                                    RQ 18/Jan/84
   14
   15                                   REFA    SYSGLOBALS,LOADER,STACKFUDGE,ASM_CACHE_ON
   16                                   REFR    FS_FCLOSE,INITUNITS_NOISR,INITLOAD_INITLOAD
   17                                   LMODE   FS_FCLOSE,INITUNITS_NOISR,INITLOAD_INITLOAD
   18                                   LMODE   ASM_CACHE_ON
   19                                   SMODE   ESCAPE
   20
   21                           *THE FOLLOWING OFFSETS ARE RELATIVE TO SYSGLOBALS(A5)
   22
   23                   FFFF FFFE ESCAPECODE      EQU SYSGLOBALS-2          (A5)
   24                   FFFF FFFA FILELISTPTR     EQU SYSGLOBALS-6          (A5)
   25                   FFFF FFF6 RECOVERBLOCK    EQU SYSGLOBALS-10         (A5)
   26                   FFFF FFF2 HEAPPOINTER     EQU SYSGLOBALS-14         (A5)
   27                   FFFF FFEE HEAPBASE        EQU SYSGLOBALS-18         (A5)
   28                   FFFF FFEA IORESULT        EQU SYSGLOBALS-22         (A5)
   29                   FFFF FFC2 INTERRUPTTABLE  EQU SYSGLOBALS-62         (A5) ADDRESS OF INTERRUPTTABLE[1..7]
   30                   FFFF FFBE ENDISRHOOK      EQU SYSGLOBALS-66         (A5) ADDRESS OF END OF ISR ROUTINE
   31                   FFFF FEEC DEBUGGER        EQU SYSGLOBALS-276        (A5) DEBUGGER HOOK
   32                   FFFF FEE4 CLEARIOHOOK     EQU SYSGLOBALS-284        (A5) CLEAR I/O HOOK
   33                   FFFF FFBA SYSDEFS         EQU LOADER-70             (A5) LINKED LIST OF PERMANT PROGRAMS
   34                   0000 0000 NIL             EQU 0
   35
   36                   FFFF FB00 HIGHMEM         EQU $FFFFFB00   LEAVES ROOM FOR VECTORS, MONITOR STUFF, ETC.
   37                   FFFF FF94 TRAPOVECTOR     EQU $FFFFFF94   LOCATION OF EXCEPTION VECTOR FOR TRAP #0
   38                             *
   39                   FFFF FF9A LEVEL7V         EQU $FFFFFF9A   NMI
   40                   FFFF FF22 LEVEL7DV        EQU $FFFFFF22   LEVEL 7 DEVICE VECTOR                   (rdq)
   41                   FFFF FF34 KBDRESETV       EQU $FFFFFF34   KEY BOARD <SHIFT PAUSE VECTOR>
   42                   FFFF FF2E FHIVECTOR       EQU $FFFFFF2E   KEY BOARD LEVEL 7 TIMER INTERRUPT       (rdq)
   43                             *
   44                   FFFF FDCE LOWMEM          EQU $FFFFFDCE   LOCATION IN BOOT ROM OF LOWEST RAM
   45                   0000 01A0 RTN_TO_MONITOR  EQU $1A0        ENTRY POINT IN BOOT ROM
   46                   0053 8000 G_OFF_MEM       EQU $538000     GRAPHICS OFF
   47                   0051 2000 ALPHA_MEM       EQU $512000     ALPHA MEMORY
   48                             *
   49                   005B 0000 P_STATUS        EQU $5B0000     PARITY STATUS REG                JS 9/12/83
   50                             *
   51                             *
   52                             * MAGIC NUMBERS, see alse FILES ASM, DEBUGGER and INITBUG
   53                             *
   54                   FFFF FB00 ERR_INFO        EQU HIGHMEM
   55                   FFFF FB00 FAULT_ADDR      EQU HIGHMEM     PARITY ERROR ADDRESS
   56                             *                                68000
   57                   FFFF FB00 BE_SSW          EQU     HIGHMEM         SPECIAL STATUS WORD
   58                   FFFF FB02 BE_FAULT_ADDR   EQU     BE_SSW+2        FAULT ADDRESS
```

```
PAGE   2 [3.0] 12/26/84 20:59:24  ASSEMBLY OF POWERUP.TEXT                                  *** File name: POWERUP ***

   59                   FFFF FB06 BE_INSTR        EQU     BE_FAULT_ADDR+4 INSTRUCTION BUFFER
   60                             *
   61                             *                                68010
   62                             *BE_SSW          2 BYTES                 VECTOR TYPE 1000
   63                             *BE_FAULT_ADDR   4 BYTES
   64                   FFFF FB06 BE_PAD1_10      EQU     BE_FAULT_ADDR+4
   65                   FFFF FB08 BE_DATA0_10     EQU     BE_PAD1_10+2    DATA INPUT BUFFER
   66                   FFFF FB0A BE_PAD2_10      EQU     BE_DATA0_10+2
   67                   FFFF FB0C BE_DATA1_10     EQU     BE_PAD2_10+2    DATA OUTPUT BUFFER
   68                   FFFF FB0E BE_PAD3_10      EQU     BE_DATA1_10+2
   69                   FFFF FB10 BE_INSTR_10     EQU     BE_PAD3_10+2    INSTRUCTION BUFFER
   70                   FFFF FB12 BE_MISC_10      EQU     BE_INSTR_10+2   16 WORDS
   71                   FFFF FB32 BE_END_10       EQU     BE_MISC_10+32
   72                             *
   73                             *                                68020
   74                   FFFF FB00 ERR_PC          EQU     HIGHMEM         VECTOR TYPE 0010
   75                             *ERR_PC          4 BYTES                 VECTOR TYPE 1001
   76                   FFFF FB04 ERR_WRD1        EQU     ERR_PC+4
   77                   FFFF FB06 ERR_WRD2        EQU     ERR_WRD1+2
   78                   FFFF FB08 ERR_EA          EQU     ERR_WRD2+2      EVALUATED ADDRESS
   79                             *BE_SSW          4 BYTES                 VECTOR TYPE 1010 (SHORT BUS ERROR)
   80                   FFFF FB04 BE_IPSC         EQU     BE_SSW+4        I PIPE C
   81                   FFFF FB06 BE_IPSB         EQU     BE_IPSC+2       I PIPE B
   82                   FFFF FB08 BE_PAD2_20      EQU     BE_IPSB+2
   83                   FFFF FB0C BE_FAULT_ADDR20 EQU     BE_PAD2_20+4
   84                   FFFF FB10 BE_DATA_20      EQU     BE_FAULT_ADDR20+4       DATA BUFFER
   85                   FFFF FB14 BE_MISC5_20     EQU     BE_DATA_20+4    4 BYTES
   86                   FFFF FB18 BE_END_S_20     EQU     BE_MISC5_20+4
   87                             *
   88                             *BE_SSW          4 BYTES                 VECTOR TYPE 1011 (LONG BUS ERROR)
   89                             *BE_IPSC         2 BYTES
   90                             *BE_IPSB         2 BYTES
   91                             *BE_PAD          4 BYTES
   92                             *BE_FAULT_ADDR   4 BYTES
   93                             *BE_DATA0        4 BYTES
   94                   FFFF FB14 BE_PAD3_20      EQU     BE_DATA_20+4            16 BYTES
   95                   FFFF FB24 BE_DATA1_20     EQU     BE_PAD3_20+16
   96                   FFFF FB28 BE_MISC20_20    EQU     BE_DATA1_20+4           44 BYTES
   97                   FFFF FB54 BE_END_L_20     EQU     BE_MISC20_20+44
   98                   FFFF FB54 BE_END          EQU     BE_END_L_20
   99                             *-------------------------------------------------------------------------------
  100                   FFFF FB54 EXCP_STATUS     EQU BE_END
  101                   FFFF FB56 EXCP_PC         EQU EXCP_STATUS+2
  102                   FFFF FB5A EXCP_VOFFSET    EQU EXCP_PC+4           VECTOR WORD FOR 680xx           (rdq)
  103                   FFFF FB5C EXCP_LINE       EQU EXCP_VOFFSET+2                                      (rdq)
  104                   FFFF FB60 LASTLINE        EQU EXCP_LINE+4
  105                   FFFF FB64 ESCAPE          EQU LASTLINE+4
  106                   FFFF FB6A PCTEMP          EQU ESCAPE+6
  107                   FFFF FB6E SRTEMP          EQU PCTEMP+4
  108                   FFFF FB70 INITSTACK       EQU SRTEMP+2
  109                   FFFF FB74 INITPC          EQU INITSTACK+4
  110                   FFFF FB78 INITRECOVER     EQU INITPC+4
  111                   FFFF FB7C G_DOLLAR        EQU INITRECOVER+4
  112                   FFFF FB80 CTL_RESETV      EQU G_DOLLAR+4          KEY BOARD <CONTROL><SHIFT><PAUSE> VECTOR
  113                   FFFF FB86 DEBUGESCAPE     EQU CTL_RESETV+6
  114                   FFFF FB8C BESPTEMP        EQU DEBUGESCAPE+6       USED IN IGNOREBUS
  115                   FFFF FB90 ALPHAFLAG       EQU BESPTEMP+4
```

```
                FFFF FB91 GRAPHICSFLAG    EQU ALPHAFLAG+1
                FFFF FB92 GRAPHICSBASE    EQU GRAPHICSFLAG+1
                FFFF FB96 INITSR          EQU GRAPHICSBASE+4
                FFFF FB98 M68KTYPE        EQU INITSR+2     PROCESSOR TYPE 0=68000 else 680xx       (rdq)
                FFFF FB99 MSYSFLAGS       EQU M68KTYPE+1   MORE SYSFLAGS BIT 0 = CACHE PRESENT/ABSENT
                FFFF FB9A FLTPTHDW        EQU MSYSFLAGS+1
                FFFF FB9B FILLER          EQU FLTPTHDW+1   UNUSED

                0000 4EF9 JMP     EQU $4EF9                       LONG ABSOLUTE JMP OPCODE


 00000000                         RORG 0

                0000 0000 ASM_INITVECTS   EQU *                   MAIN PROGRAM POWER UP LOCATION
 00000000 4238 FB9A               CLR.B    FLTPTHDW        SHOW NO FLOATING POINT HARDWARE

 00000004 70FF                    MOVEQ    #$FFFFFFFF,D0           HIGHEST POSSIBLE ADDRESS
 00000006 2B40 FFFA               MOVE.L   D0,FILELISTPTR(A5)
 0000000A 31FC 4E75               MOVE.W   #$4E75,DEBUGESCAPE
          FB86
 00000010 42B8 FB60               CLR.L    LASTLINE
 00000014 2B7C 0000               MOVE.L   #ENDISR,ENDISRHOOK(A5)  SET UP ORDINARY RETURN FROM ISR's
          030E FFBE

 0000001C 4EBA 05EC               jsr      init_timer      checks for and turns on timer   (rdq)

                          *                                TURN OFF GRAPHICS
 00000020 207C 0053               MOVEA.L  #G_OFF_MEM,A0   GET MEM ADDRESS
          8000
 00000026 21C8 FB92               MOVE.L   A0,GRAPHICSBASE SAVE IT IN GLOBAL AREA
 0000002A 4238 FB91               CLR.B    GRAPHICSFLAG    MARK IT TURNED OFF

 0000002E 43FA 0172               LEA      IGNOREBUS,A1    SET BUS ERROR VECTOR
 00000032 21C9 FFFC               MOVE.L   A1,-4
 00000036 31FC 4EF9               MOVE.W   #JMP,-6
          FFFA
 0000003C 487A 000A               PEA      GRPH1
 00000040 21CF FB8C               MOVE.L   SP,BESPTEMP
 00000044 3010                    MOVE     (A0),D0         TURN OFF GRAPHICS
 00000046 588F                    ADDQ.L   #4,SP
                0000 0048 GRPH1   EQU      *
 00000048 11FC 0001               MOVE.B   #1,ALPHAFLAG    MARK ALPHA TURNED ON
          FB90

                          *
                          *    TURN ON CACHE IF WE HAVE ONE
                          *
                005F 400E CACHE_CTL EQU   $5F400E
 0000004E 4238 FB99               CLR.B    MSYSFLAGS       CLEAR FLAGS BYTE
 00000052 487A 001A               PEA      CACHE1
 00000056 21CF FB8C               MOVE.L   SP,BESPTEMP
 0000005A 4A79 005F               TST.W    CACHE_CTL       TEST
          400E
 00000060 588F                    ADDQ.L   #4,SP
 00000062 08F8 0000               BSET     #0,MSYSFLAGS    CACHE EXISTS
          FB99
 00000068 4EB9 0000               JSR      ASM_CACHE_ON
          0000
```



```
                0000 006E CACHE1  EQU      *
                          *
                          *    TURN ON PARITY BOARDS IF ANY                              JS 9/12/83
                          *                                                              JS 9/12/83
 0000006E 487A 0010               PEA      PCHK1
 00000072 21CF FB8C               MOVE.L   SP,BESPTEMP
 00000076 33FC 0001               MOVE.W   #1,P_STATUS                                   JS 9/12/83
          005B 0000
 0000007E 588F                    ADDQ.L   #4,SP
                0000 0080 PCHK1   EQU      *                                             JS 9/12/83
                          *
 00000080 41F8 FFC4               LEA      $FFFFFFC4,A0            ADDRESS PAST INTERRUPT VECTOR LEVEL 1
 00000084 43FA 01E6               LEA      INTERRUPT,A1
 00000088 7006                    MOVEQ    #6,D0                   HANDLE LEVELS 1 THRU 6
 0000008A 2109          I_LOOP    MOVE.L   A1,-(A0)                MOVE JMP TO INTERRUPT VECTOR
 0000008C 313C 4EF9               MOVE.W   #JMP,-(A0)
 00000090 5340                    SUBQ     #1,D0
 00000092 66F6                    BNE      I_LOOP

 00000094 41F8 FF22               LEA      LEVEL7DV,A0     SETUP LEVEL 7 DEVICE VECTOR             (rdq)
 00000098 30FC 4EF9               MOVE.W   #JMP,(A0)+                                      (rdq)
 0000009C 2089                    MOVE.L   A1,(A0)                                        (rdq)
                          * INIT VECTORS FOR HANDLING <SHIFT PAUSE> AND <CONTROL><SHIFT><PAUSE>
 0000009E 31FC 4EF9               MOVE.W   #JMP,LEVEL7V
          FF9A
 000000A4 21F8 01B0               MOVE.L   $1B0,LEVEL7V+2
          FF9C
 000000AA 31FC 4EF9               MOVE.W   #JMP,KBDRESETV
          FF34
 000000B0 21FC 0000               MOVE.L   #RESET_ISR,KBDRESETV+2
          04AC FF36
 000000B8 31FC 4EF9               MOVE.W   #JMP,CTL_RESETV
          FB80
 000000BE 21FC 0000               MOVE.L   #TRYMONITOR,CTL_RESETV+2
          04CA FB82
                          *
 000000C6 41F8 FF2E               LEA      FHIVECTOR,A0    LEVEL 7 TIMER INTERRUPT VECTOR          (rdq)
 000000CA 30FC 4EF9               MOVE.W   #JMP,(A0)+                                      (rdq)
 000000CE 2089                    MOVE.L   A1,(A0)                                        (rdq)

 000000D0 41FA 0342               LEA      ESC,A0                  MOVE ESCAPE LINKAGE TO HIGH MEMORY
 000000D4 31FC 4EF9               MOVE.W   #JMP,ESCAPE
          FB64
 000000DA 21C8 FB66               MOVE.L   A0,ESCAPE+2

 000000DE 2038 FF96               MOVE.L   TRAPOVECTOR+2,D0        GET DEFAULT ENTRY POINT FOR TRAP #0
 000000E2 C0BC 00FF               AND.L    #$00FFFFFF,D0           TEST FOR MONITOR
          FFFF
 000000E8 B0BC 0088               CMP.L    #$00880000,D0
          0000
 000000EE 6C0E                    BGE.S    L0
 000000F0 31FC 4EF9               MOVE.W   #JMP,TRAPOVECTOR        SET UP EXCEPTION VECTOR FOR
          FF94
 000000F6 41FA 03A0               LEA      P_BREAK,A0              PASCAL LINE HEADERS
 000000FA 21C8 FF96               MOVE.L   A0,TRAPOVECTOR+2

                          *==========
```

```
PAGE   5 [3.0] 12/26/84 20:59:24  ASSEMBLY OF POWERUP.TEXT                                  *** File name: POWERUP ***

  211
  212   000000FE 4CBA 0006 L0      MOVEM.W ESCAPE_PROTO,D1-D2      GET 6 BYTE MOVEQ #ESCCODE, JMP ESCAPE
                   03A6
  213   00000104 41F8 0000         LEA     0,A0                    START FROM TOP OF RAM
  214   00000108 303C 4EF9         MOVE.W  #JMP,D0                 $4EF9   (LONG ABS JUMP)
  215
  216   0000010C 43FA 0270         LEA     BUS_ERR,A1              SET UP BUS ERROR TRAP
  217   00000110 2109              MOVE.L  A1,-(A0)                ADDRESS IN LAST 4 BYTES
  218   00000112 3100              MOVE.W  D0,-(A0)                LONG JUMP IN FIRST 2 BYTES
  219
  220   00000114 43FA 025C         LEA     ADD_ERR,A1              SET UP ADDRESS ERROR TRAP
  221   00000118 2109              MOVE.L  A1,-(A0)                ADDRESS IN LAST 4 BYTES
  222   0000011A 3100              MOVE.W  D0,-(A0)                LONG JUMP IN FIRST 2 BYTES
  223
  224   0000011C 76F3              MOVEQ   #-13,D3                 ILLEGAL INSTRUCTION, ESCAPE(-13)
  225   0000011E 48A0 7000         MOVEM.W D1-D3,-(A0)             MOVE 'moveq, short jump esc'
  226
  227   00000122 76FB              MOVEQ   #-5,D3                  SIMILAR FOR DIVIDE BY ZERO
  228   00000124 48A0 7000         MOVEM.W D1-D3,-(A0)
  229
  230   00000128 76F8              MOVEQ   #-8,D3                  SIMILAR FOR CHK EXCEPTION
  231   0000012A 48A0 7000         MOVEM.W D1-D3,-(A0)
  232
  233   0000012E 76FC              MOVEQ   #-4,D3                  SIMILAR FOR TRAPV
  234   00000130 48A0 7000         MOVEM.W D1-D3,-(A0)
  235
  236   00000134 76F2              MOVEQ   #-14,D3                 SIMILAR FOR PRIVILEGE VIOLATION
  237   00000136 48A0 7000         MOVEM.W D1-D3,-(A0)
  238
  239   0000013A 5D88              SUBQ.L  #6,A0                   SKIP TRACE
  240
  241   0000013C 76F3              MOVEQ   #-13,D3                 ILLEGAL INSTRUCTION (OPS A,F), ESCAPE(-13)
  242   0000013E 48A0 7000         MOVEM.W D1-D3,-(A0)             MOVE 'moveq, short jump esc'
  243   00000142 48A0 7000         MOVEM.W D1-D3,-(A0)             MOVE 'moveq, short jump esc'
  244
  245   00000146 41F8 FF94         LEA     $FFFFFF94,A0            CONTINUE WITH TRAP VECTORS
  246
  247   0000014A 43FA 00EC         LEA     LINKA6,A1               TRAP 1, LINK A6 EMULATOR
  248   0000014E 2109              MOVE.L  A1,-(A0)
  249   00000150 3100              MOVE.W  D0,-(A0)
  250
  251   00000152 76FE              MOVEQ   #-2,D3                  TRAP 2, STACK OVERFLOW
  252   00000154 48A0 7000         MOVEM.W D1-D3,-(A0)
  253
  254   00000158 76F6              MOVEQ   #-10,D3                 TRAP 3, I/O RESULT NOT ZERO
  255   0000015A 48A0 7000         MOVEM.W D1-D3,-(A0)
  256
  257   0000015E 76FC              MOVEQ   #-4,D3                  TRAP 4, INTEGER OVERFLOW
  258   00000160 48A0 7000         MOVEM.W D1-D3,-(A0)
  259
  260   00000164 76FB              MOVEQ   #-5,D3                  TRAP 5, INTEGER DIVIDE BY ZERO
  261   00000166 48A0 7000         MOVEM.W D1-D3,-(A0)
  262
  263   0000016A 76F7              MOVEQ   #-9,D3                  TRAP 6, CASE STATEMENT ERROR
  264   0000016C 48A0 7000         MOVEM.W D1-D3,-(A0)
  265
  266   00000170 76F8              MOVEQ   #-8,D3                  TRAP 7, VALUE RANGE ERROR
```

```
PAGE   6 [3.0] 12/26/84 20:59:24  ASSEMBLY OF POWERUP.TEXT                                  *** File name: POWERUP ***

  267   00000172 48A0 7000         MOVEM.W D1-D3,-(A0)
  268
  269   00000176 76FD              MOVEQ   #-3,D3                  TRAP 8, NIL POINTER REFERENCE
  270   00000178 48A0 7000         MOVEM.W D1-D3,-(A0)
  271
  272   0000017C 43FA 0072         LEA     NLGOTO,A1               TRAP 9, NON LOCAL GOTO
  273   00000180 2109              MOVE.L  A1,-(A0)
  274   00000182 3100              MOVE.W  D0,-(A0)
  275
  276   00000184 43FA 02A6         LEA     ESCN,A1                 TRAP 10, ESCAPE N
  277   00000188 2109              MOVE.L  A1,-(A0)
  278   0000018A 3100              MOVE.W  D0,-(A0)
  279                        *                                      (rdq)
  280   0000018C 5D88              SUBQ.L  #6,A0                   skip over TRAP 11 (done by ASM_POWERUP)
  281                        *
  282   0000018E 76EB              MOVEQ   #-21,D3                 TRAP 12, UNASSIGNED
  283   00000190 48A0 7000         MOVEM.W D1-D3,-(A0)
  284
  285   00000194 76EB              MOVEQ   #-21,D3                 TRAP 13, UNASSIGNED
  286   00000196 48A0 7000         MOVEM.W D1-D3,-(A0)
  287
  288   0000019A 76EB              MOVEQ   #-21,D3                 TRAP 14, UNASSIGNED
  289   0000019C 48A0 7000         MOVEM.W D1-D3,-(A0)
  290
  291   000001A0 4E75              RTS                             END OF VECTOR SET UP, POWER UP
  292                        *-------(rdq)-----------------------------------------------------------
  293              0000 01A2 IGNOREBUS EQU   *
  294   000001A2 4A38 FB98         TST.B   M68KTYPE
  295   000001A6 6606              BNE.S   IGNBUS1
  296   000001A8 DFFC 0000         ADDA.L  #8,SP                   THROW AWAY STACK INFO FROM BUS ERROR
                   0008
  297   000001AE 46D7      IGNBUS1 MOVE  (SP),SR                   FIX SR SO SP WILL BE CORRECT
  298   000001B0 2E78 FB8C         MOVEA.L BESPTEMP,SP
  299   000001B4 4E75              RTS                             RETURN TO RECOVERY POINT
  300                        *=======================================================================
  301              0000 01B6 ASM_CLOSEFILES EQU *
  302   000001B6 20EF 0004         MOVEA.L 4(SP),A0        EVENTUAL SP
  303   000001BA 226D FFFA         MOVEA.L FILELISTPTR(A5),A1
  304   000001BE B3C8              CMPA.L  A0,A1
  305   000001C0 642A              BCC.S   ALLDONE
  306   000001C2 B3CF              CMPA.L  SP,A1
  307   000001C4 6526              BCS.S   ALLDONE
  308   000001C6 2B69 0004         MOVE.L  4(A1),FILELISTPTR(A5)
                   FFFA
  309   000001CC 42A9 0004         CLR.L   4(A1)           FLAG UNINITIALIZED
  310   000001D0 2F2D FFEA         MOVE.L  IORESULT(A5),-(SP)
  311   000001D4 3F2D FFFE         MOVE.W  ESCAPECODE(A5),-(SP)
  312   000001D8 4851              PEA     (A1)
  313   000001DA 4267              CLR.W   -(SP)           NORMAL CLOSE
  314   000001DC 4EB9 0000         JSR     FS_FCLOSE
                   0000
  315   000001E2 3B5F FFFE         MOVE.W  (SP)+,ESCAPECODE(A5)
  316   000001E6 2B5F FFEA         MOVE.L  (SP)+,IORESULT(A5)
  317   000001EA 60CA              BRA.S   ASM_CLOSEFILES
  318   000001EC 2E9F      ALLDONE MOVE.L  (SP)+,(SP)
  319   000001EE 4E75              RTS
  320
```

```
000001F0 301F      NLGOT0  MOVE.W  (SP)+,D0          SAVE STATUS REG
000001F2 205F              MOVEA.L (SP)+,A0
000001F4 4A38 FB98         TST.B   M68KTYPE                                    (rdq)
000001F8 6702              BEQ.S   NLGOT0A                                     (rdq)
000001FA 548F              ADDQ.L  #2,SP             POP VECTOR WORD           (rdq)
000001FC 46C0      NLGOT0A MOVE    D0,SR             RESTORE USER MODE
000001FE 3218              MOVE.W  (A0)+,D1          STATIC DELTA
00000200 6C04              BGE.S   NLGOT01
00000202 2C78 FB70         MOVEA.L INITSTACK,A6      DEST IS MAIN PROG
00000206 6F08      NLGOT01 BLE.S   NLGOT02
00000208 2C6E 0008 NLGOT02 MOVEA.L 8(A6),A6
0000020C 5341              SUBQ.W  #1,D1
0000020E 6EF8              BGT.S   NLGOT02
00000210 2210      NLGOT03 MOVE.L  (A0),D1           DESTINATION DELTA
00000212 4870 1800         PEA     0(A0,D1.L)        COMPUTE RETURN ADDRESS
00000216 3430 1802         MOVE.W  2(A0,D1.L),D2     SP DELTA FROM A6
0000021A 43F6 2000         LEA     0(A6,D2.W),A1     EVENTUAL SP
0000021E B3ED FFF6 NLGOT04 CMPA.L  RECOVERBLOCK(A5),A1 POP OFF TRY RECOVER BLOCKS
00000222 630C              BLS.S   NLGOT05           ABOVE THE EVENTUAL SP
00000224 246D FFF6         MOVEA.L RECOVERBLOCK(A5),A2
00000228 2B6A 0008         MOVE.L  8(A2),RECOVERBLOCK(A5)
         FFF6
0000022E 60EE              BRA.S   NLGOT04
00000230 2F09      NLGOT05 MOVE.L  A1,-(SP)
00000232 4EBA FF82         JSR     ASM_CLOSEFILES    CLOSE FILES BEING POPPED OFF
00000236 4E75              RTS

00000238 301F      LINKA6  MOVE    (SP)+,D0          STATUS
0000023A 205F              MOVEA.L (SP)+,A0          PC
0000023C 4A38 FB98         TST.B   M68KTYPE                                    (rdq)
00000240 6702              BEQ.S   LINKA6A                                     (rdq)
00000242 548F              ADDQ.L  #2,SP             POP VECTOR WORD           (rdq)
00000244 46C0      LINKA6A MOVE    D0,SR             RETURN TO ORIGINAL MODE
00000246 2F0E              MOVE.L  A6,-(SP)          EMULATE LINK A6
00000248 2C4F              MOVEA.L SP,A6
0000024A 264F              MOVEA.L SP,A3             COPY FOR SP CALC.
0000024C D6D8              ADDA.W  (A0)+,A3          PRE CALCULATE NEW SP

0000024E 0800 000D         BTST    #13,D0            WHICH MODE?
00000252 6706              BEQ.S   UMODE

00000254 45ED 7FFE SMODE   LEA     32766(A5),A2
00000258 6004              BRA.S   SCHECK
0000025A 246D FFF2 UMODE   MOVEA.L HEAPPOINTER(A5),A2

0000025E 43EB 0000 SCHECK  LEA     -STACKFUDGE(A3),A1

00000262 B3CA              CMPA.L  A2,A1   CHECK STACK OVERFLOW
00000264 6304              BLS.S   STACKOV
00000266 2E4B              MOVEA.L A3,SP   NOW SET NEW SP
00000268 4ED0              JMP     (A0)

0000026A 4E42      STACKOV TRAP    #2                SIGNAL STACK OVERFLOW
```



```
                    ********************************************************************************
                    *                    INTERRUPT POLLING ROUTINE: LEVELS 1-7 (rdq)               *
                    ********************************************************************************

                    *
                    * 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 1-7                                    (rdq)
                    *
          0000 026C INTERRUPT       EQU *

0000026C 48E7 FFFE          MOVEM.L D0-D7/A0-A6,-(SP)       SAVE CONTEXT, 64 BYTES
00000270 2A78 FB7C          MOVEA.L G_DOLLAR,A5             SET UP THE PASCAL GLOBAL ENVIRONMENT
00000274 2F2D FFEA          MOVE.L  IORESULT(A5),-(SP)      IORESULT IS PART OF THE CONTEXT
00000278 3F2D FFFE          MOVE.W  ESCAPECODE(A5),-(SP)    THE ESCAPE CODE IS PART OF THE CONTEXT

0000027C 2F2D FFF6          MOVE.L  RECOVERBLOCK(A5),-(SP)  SET UP A TRY/RECOVER BLOCK
00000280 487A 00BE          PEA     RECOVER
00000284 2B4F FFF6          MOVE.L  SP,RECOVERBLOCK(A5)

00000288 40C0               MOVE    SR,D0                   FETCH THE CURRENT INTERRUPT LEVEL
0000028A EC48               LSR     #8-2,D0                 POSITION IN THE LOWER BYTE, MULTIPLIED BY 4
0000028C C07C 001C          AND.W   #$1C,D0                 MASK OUT THE OTHER BITS
00000290 41ED FFC2          LEA     INTERRUPTTABLE(A5),A0
00000294 2070 00FC          MOVEA.L -4(A0,D0),A0            POINT TO THE FIRST ISRIB FOR THIS LEVEL

00000298 2008       PLOOP   MOVE.L  A0,D0                   SET THE CONDITION CODES
0000029A 6722               BEQ.S   NOISR                   BRANCH IF THE POINTER IS NILL
0000029C 2248               MOVEA.L A0,A1                   USE A1 FOR SCANNING THE ISRIB
0000029E 2459               MOVEA.L (A1)+,A2                INTERRUPT REGISTER ADDRESS
000002A0 1012               MOVE.B  (A2),D0                 INTERRUPT REGISTER CONTENTS
000002A2 C019               AND.B   (A1)+,D0                INTERRUPT REGISTER MASK
000002A4 B019               CMP.B   (A1)+,D0                THIS SOURCE REQUESTING AN INTERRUPT?
000002A6 6610               BNE.S   NEXT                    BRANCH IF NOT

000002A8 4259               CLR     (A1)+                   CLEAR THE CHAIN FLAG
000002AA 2F08               MOVE.L  A0,-(SP)                SAVE THE POINTER TO THIS ISRIB
000002AC 2F08               MOVE.L  A0,-(SP)                ALSO, THE ISR GETS IT AS A PARAMETER

000002AE 6164               BSR.S   CALLPROC                CALL THE ISR

000002B0 205F               MOVEA.L (SP)+,A0                RESTORE THE POINTER TO THIS ISRIB
000002B2 4A68 0006          TST     CHAINFLAG(A0)           DID THIS ISR CHOOSE TO SERVICE THE INTERRUPT?
000002B6 6742               BEQ.S   RESTORE                 BRANCH IF SO; OTHERWISE...

000002B8 2068 0010  NEXT    MOVEA.L LINK(A0),A0             POINT TO THE NEXT ISRIB IN THE LINKED LIST
000002BC 60DA               BRA     PLOOP                   AND POLL IT's associated interrupt bit
```

```
  432    000002BE 4000      NOISR   MOVE    SR,D0                   CHECK INTERRUPT LEVEL       JS 9/12/83
  433    000002C0 E048              LSR     #8,D0                                               JS 9/12/83
  434    000002C2 0240 0007         ANDI    #7,D0                   MASK ALL BUT LEVEL BITS     JS 9/12/83
  435    000002C6 0C40 0007         CMPI    #7,D0                   IS IT AN NMI ?              JS 9/12/83
  436    000002CA 6626              BNE.S   NOISRB                  NO, TREAT AS BEFORE         JS 9/12/83
  437    000002CC 4EBA 02AE         JSR     P_CHECK                 PARITY ERROR?               JS 9/12/83
  438    000002D0 6720              BEQ.S   NOISRB                  NO, SOMETHING ELSE          JS 9/12/83
  439    000002D2 588F              ADDQ.L  #4,SP                   POP ISR RECOVER ADDR        JS 9/12/83
  440    000002D4 2B5F FFF6         MOVE.L  (SP)+,RECOVERBLOCK(A5)  RESTORE OLD RECOVERBLOCK    JS 9/12/83
  441    000002D8 3B5F FFFE         MOVE.W  (SP)+,ESCAPECODE(A5)    ESCAPECODE AND IORESULT     JS 9/12/83
  442    000002DC 2B5F FFEA         MOVE.L  (SP)+,IORESULT(A5)      FROM USER CONTEXT           JS 9/12/83
  443    000002E0 4CDF 7FFF         MOVEM.L (SP)+,D0-D7/A0-A6       AND LOOK AS IF WE DID A     JS 9/12/83
  444    000002E4 3B7C FFE4         MOVE.W  #-28,ESCAPECODE(A5)     ESCAPE(-28) FROM THE        JS 9/12/83
                  FFFE
  445    000002EA 42B8 FB00         CLR.L   FAULT_ADDR              USER PROGRAM                JS 9/12/83
  446    000002EE 6000 013C         BRA     ESCN                                                JS 9/12/83
  447
  448    000002F2 2F08      NOISRB  MOVE.L  A0,-(SP)                NILL POINTER
  449    000002F4 4EB9 0000         JSR     INITUNITS_NOISR
                  0000
  450
  451
  452    000002FA 588F      RESTORE ADDQ.L  #4,SP                   POP OFF THE RECOVER BLOCK ADDRESS
  453    000002FC 2B5F FFF6 RECOV_1 MOVE.L  (SP)+,RECOVERBLOCK(A5)  RESTORE THE ORIGINAL RECOVER BLOCK
  454    00000300 3B5F FFFE         MOVE.W  (SP)+,ESCAPECODE(A5)    RESTORE THE ORIGINAL ESCAPE CODE
  455    00000304 2B5F FFEA         MOVE.L  (SP)+,IORESULT(A5)      RESTORE IORESULT
  456    00000308 206D FFBE         MOVEA.L ENDISRHOOK(A5),A0       CALL ISR HOOK TO ALLOW MULTI-TASKING
  457    0000030C 4ED0              JMP     (A0)
  458
  459    0000030E 4CDF 7FFF ENDISR  MOVEM.L (SP)+,D0-D7/A0-A6       RESTORE THE ORIGINAL CONTEXT
  460    00000312 4E73              RTE                             END OF INTERRUPT SERVICE
  461
  462              0000 0314 CALLPROC EQU *                         PROCEDURE CALL
  463    00000314 2059              MOVEA.L (A1)+,A0                PROC ADDRESS
  464    00000316 2019              MOVE.L  (A1)+,D0                STATIC LINK
  465    00000318 6706              BEQ.S   CALLIT                  SKIP IF THERE IS NO STATIC LINK
  466    0000031A 225F                MOVEA.L (SP)+,A1              SHUFFLE RETURN ADDRESS
  467    0000031C 2F00                MOVE.L  D0,-(SP)              PUSH STATIC LINK ON THE STACK
  468    0000031E 2F09                MOVE.L  A1,-(SP)
  469    00000320 4ED0      CALLIT  JMP     (A0)                    CALL THE PROCEDURE
  470
  471
  472                       *
  473                       * RECOVER BLOCK ROUTINES TO CATCH ESCAPES FROM ISR'S
  474                       *
  475    00000322 7046      RECOV_2 MOVEQ   #70,D0                  SETUP TO ENTER DEBUGGER          (rdq)
  476    00000324 31F7 0000         MOVE.W  0(SP,D0),SRTEMP         SAVE THE STATUS REGISTER         (rdq)
                  FB6E
  477    0000032A 3FBC 2100         MOVE.W  #$2100,0(SP,D0)         DUMMY STATUS                     (rdq)
                  0000
  478    00000330 2238 FB6A         MOVE.L  PCTEMP,D1               SAVE THE TARGET ADDRESS          (rdq)
  479    00000334 21F7 0002         MOVE.L  2(SP,D0),PCTEMP         SWITCH PCTEMP                    (rdq)
                  FB6A
  480    0000033A 2F81 0002         MOVE.L  D1,2(SP,D0)                                              (rdq)
  481    0000033E 60BC              BRA.S   RECOV_1
  482
```



```
  483    00000340 0C6D FFEA RECOVER CMPI.W  #-22,ESCAPECODE(A5)     TEST FOR DEBUGGER CALL
                  FFFE
  484    00000346 67DA              BEQ     RECOV_2
  485    00000348 0C6D FFE4         CMPI.W  #-28,ESCAPECODE(A5)     CHECK FOR PARITY ERROR    JS 9/12/83
                  FFFE
  486    0000034E 6708              BEQ.S   RECOVRX                 SAME TREATMENT AS STOP    JS 9/12/83
  487    00000350 0C6D FFEC         CMPI.W  #-20,ESCAPECODE(A5)     TEST FOR STOP KEY
                  FFFE
  488    00000356 66A4              BNE     RECOV_1
  489
  490                       *        STOP KEY PRESSED               SIMULATE AN ESCAPE(-20)
  491    00000358 2B5F FFF6 RECOVRX MOVE.L  (SP)+,RECOVERBLOCK(A5)  RESTORE THE ORIGINAL RECOVER BLOCK
  492    0000035C 548F              ADDQ.L  #2,SP                   GET RID OF SAVED ESCAPECODE
  493    0000035E 2B5F FFEA         MOVE.L  (SP)+,IORESULT(A5)      RESTORE IORESULT
  494    00000362 41FA 000C         LEA     STOP,A0
  495    00000366 2F48 003E RECOVERA MOVE.L A0,62(SP)     REDIRECT INTERRUPTED PROGRAM TO STOP    (rdq)
  496    0000036A 206D FFBE         MOVEA.L ENDISRHOOK(A5),A0       CALL ISR HOOK TO ALLOW MULTI-TASKING
  497    0000036E 4ED0              JMP     (A0)
  498
  499    00000370 4E4A      STOP    TRAP    #10
  500
  501    00000372 2A78 FB7C ADD_ERR MOVEA.L G_DOLLAR,A5             FIX A5
  502    00000376 3B7C FFF5         MOVE.W  #-11,ESCAPECODE(A5)     SET ESCAPE CODE
                  FFFE
  503    0000037C 600A              BRA.S   BE_INFO
  504    0000037E 2A78 FB7C BUS_ERR MOVEA.L G_DOLLAR,A5             FIX A5
  505    00000382 3B7C FFF4         MOVE.W  #-12,ESCAPECODE(A5)     SET ESCAPE CODE
                  FFFE
  506    00000388 4A38 FB98 BE_INFO TST.B   M68KTYPE                                                (rdq)
  507    0000038C 6778              BEQ.S   BE_INFOA                                                (rdq)
  508                       *-(rdq)-new code for 680xx ------------------------------
  509    0000038E 31DF FB54         MOVE.W  (SP)+,EXCP_STATUS       SAVE RELEVANT DIAGNOSTIC INFO 680xx
  510    00000392 21DF FB56         MOVE.L  (SP)+,EXCP_PC
  511    00000396 31DF FB5A E680XX  MOVE.W  (SP)+,EXCP_VOFFSET      other 680xx EXCEPTIONS enter here
  512    0000039A 0838 0007         BTST    #7,EXCP_VOFFSET         ?0XX note: bit 6 is assumed 0
                  FB5A
  513    000003A0 6612              BNE.S   BE02
  514    000003A2 0838 0005         BTST    #5,EXCP_VOFFSET         00?X
                  FB5A
  515    000003A8 67C0 0096         BEQ     ESCNA                   000X either 0000 or 0001
  516    000003AC 21DF FB00         MOVE.L  (SP)+,ERR_PC            001X assumed to be 0010
  517    000003B0 6000 008E         BRA     ESCNA
  518                       *
  519    000003B4 48F8 0101 BE02    MOVEM.L A0/D0,BE_END-8  SAVE SCRATCH REGS
                  FB4C
  520    000003BA 41F8 FB00         LEA     ERR_INFO,A0     FRONT OF SAVE AREA
  521    000003BE 7002              MOVEQ   #2,D0           SET FOR 1001
  522    000003C0 0838 0004         BTST    #4,EXCP_VOFFSET         10X?
                  FB5A
  523    000003C6 6720              BEQ.S   BE04
  524    000003C8 0838 0005         BTST    #5,EXCP_VOFFSET         10?1
                  FB5A
  525    000003CE 6728              BEQ.S   BE06    IF 0 THEN HAVE  1001
  526    000003D0 7012              MOVEQ   #18,D0          1011 (LONG 68020 EXECPTION)
  527    000003D2 20DF      BE03    MOVE.L  (SP)+,(A0)+
  528    000003D4 51C8 FFFC         DBRA    D0,BE03
```

```
PAGE  11 [3.0] 12/26/84 20:59:24  ASSEMBLY OF POWERUP.TEXT                                  *** File name: POWERUP ***

  529    00000308 4CF8 0101          MOVEM.L BE_END-8,A0/D0  RESTORE SCRATCH REGS
                  FB4C
  530    0000030E 21DF FB4C          MOVE.L  (SP)+,BE_END_L_20-8     SAVE LAST 2 LONG WORDS
  531    00000312 21DF FB50          MOVE.L  (SP)+,BE_END_L_20-4
  532    00000316 6058               BRA.S   ESCNA
  533                        *
  534    00000318 7005       BE04    MOVEQ   #5,D0           SET FOR 1010
  535    0000031A 0838 0005          BTST    #5,EXCP_VOFFSET         10?0
                  FB5A
  536    0000031F 6600 0006          BNE     BE06
  537    00000323 30DF               MOVE.W  (SP)+,(A0)+     HAVE 1000 MUST BE 68010
  538    00000326 700B               MOVEQ   #11,D0
  539    00000328 20DF       BE06    MOVE.L  (SP)+,(A0)+     MOVE FROM STACK TO SAVE AREA
  540    0000032A 51C8 FFFC          DBRA    D0,BE06
  541    0000032E 4CF8 0101          MOVEM.L BE_END-8,A0/D0  RESTORE SCRATCH REGS
                  FB4C
  542    00000404 603A               BRA.S   ESCNA
  543
  544                        *-------------------------------------------------------
  545    00000406 31DF FB00 BE_INFOA MOVE.W  (SP)+,BE_SSW            SAVE RELEVANT DIAGNOSTIC INFO 68000
  546    0000040A 21DF FB02          MOVE.L  (SP)+,BE_FAULT_ADDR
  547    0000040E 31DF FB06          MOVE.W  (SP)+,BE_INSTR
  548    00000412 6018               BRA.S   ESCN                THE REST IS LIKE OTHER EXCEPTIONS
  549
  550    00000414 2F0D       ESC     MOVE.L  A5,-(SP)            SAVE A5
  551    00000416 2A6F 0004          MOVEA.L 4(SP),A5            GET CODE POINTER
  552    0000041A 3F55 0006          MOVE.W  (A5),6(SP)          SAVE CODE
  553    0000041E 2A78 FB7C          MOVEA.L G_DOLLAR,A5
  554    00000422 3B6F 0006          MOVE.W  6(SP),ESCAPECODE(A5)    SET ERROR CODE
                  FFFE
  555    00000428 2A5F               MOVEA.L (SP)+,A5            RESTORE A5
  556    0000042A 588F               ADDQ.L  #4,SP               POP SCRATCH SPACE
  557
  558    0000042C 31DF FB54 ESCN     MOVE.W  (SP)+,EXCP_STATUS       SAVE EXCEPTION DIAGNOSTIC INFO
  559    00000430 21DF FB56          MOVE.L  (SP)+,EXCP_PC
  560    00000434 4A38 FB98          TST.B   M68KTYPE                                            (rdq)
  561    00000438 6600 FF5C          BNE     E680XX                                              (rdq)
  562    0000043C 4278 FB5A          CLR.W   EXCP_VOFFSET                                        (rdq)
  563    00000440 2F08       ESCNA   MOVE.L  A0,-(SP)            SAVE A0                         (rdq)
  564    00000442 21FC FFFF          MOVE.L  #-1,EXCP_LINE       TRY TO GET LINE #
                  FFFF FB5C
  565    0000044A 2078 FB60          MOVEA.L LASTLINE,A0
  566    0000044E 0C58 4E40          CMPI.W  #$4E40,(A0)+        TRAP 0 ?
  567    00000452 6608               BNE.S   NO_LINE
  568    00000454 4278 FB5C          CLR.W   EXCP_LINE
  569    00000458 31D0 FB5E          MOVE.W  (A0),EXCP_LINE+2    FOUND LINE #
  570    0000045C 205F       NO_LINE MOVEA.L (SP)+,A0            RESTORE A0
  571    0000045E 4EB8 FB86          JSR     DEBUGESCAPE         DEBUGGER HOOK
  572    00000462 4E4B               TRAP    #11
  573    00000464 548F               ADDQ.L  #2,SP                                               (rdq)
  574    00000466 2A78 FB7C          MOVEA.L G_DOLLAR,A5         FIX A5
  575
  576                        * TRY TO DETECT CASES OF INTERRUPTED SUPERVISOR CALLS DURING USER PROGRAM
  577    0000046A 206D FFF6          MOVEA.L RECOVERBLOCK(A5),A0
  578    0000046E B1CF               CMPA.L  SP,A0
  579    00000470 6C08               BGE.S   TEST_2
```

```
PAGE  12 [3.0] 12/26/84 20:59:24  ASSEMBLY OF POWERUP.TEXT                                  *** File name: POWERUP ***

  580    00000472 08B8 0005          BCLR    #5,EXCP_STATUS          MOVE BACK TO USER MODE BECAUSE USER
                  FB54
  581    00000478 600C               BRA.S   NORMAL_RECOVER            STACK IS BELOW SUPERVISOR STACK
  582    0000047A B1F8 FB78 TEST_2   CMPA.L  INITRECOVER,A0
  583    0000047E 6606               BNE.S   NORMAL_RECOVER
  584    00000480 31F8 FB96          MOVE    INITSR,EXCP_STATUS      INITSR WAS SAVED WHEN INITRECOVER WAS
                  FB54
  585
  586              0000 0486 NORMAL_RECOVER  EQU *
  587    00000486 46F8 FB54          MOVE    EXCP_STATUS,SR          RETURN TO USER MODE
  588    0000048A 2F2D FFF6          MOVE.L  RECOVERBLOCK(A5),-(SP)  GO CLOSE ALL FILES ABOVE NEW (SP)
  589    0000048E 6100 FD26          BSR     ASM_CLOSEFILES
  590    00000492 2E6D FFF6          MOVEA.L RECOVERBLOCK(A5),SP     CUT BACK USER's stack
  591    00000496 4E75               RTS
  592
  593    00000498 55AF 0002 P_BREAK  SUBQ.L  #2,2(SP)        BACKUP TO OPCODE
  594    0000049C 21EF 0002          MOVE.L  2(SP),LASTLINE  SAVE THE PC
                  FB60
  595    000004A2 58AF 0002          ADDQ.L  #4,2(SP)        BUMP RETURN ADDRESS TO SKIP LINE NUMBER
  596    000004A6 4E73               RTE                     RETURN FROM TRAP #0
  597
  598    000004A8 4EB8 FB64 ESCAPE_PROTO    JSR     ESCAPE
```

```
                        *
                        * HANDLING LEVEL 7 KEYBOARD RESET INTERUPTS
                        *
              FF88 0024 MONITOR     EQU  $FF880024
              0042 8003 KBDSTATUS   EQU  $00428003
              0042 8001 KBDDATA     EQU  $00428001
              0042 8003 KBDCOMMAND  EQU  $00428003
                        *
000004AC 48E7 FFFE RESET_ISR  MOVEM.L D0-D7/A0-A6,-(SP)   SAVE REGISTERS
000004B0 13FC 00B2            MOVE.B #$B2,KBDCOMMAND      TURN OFF INTERRUPT
              0042 8003
000004B8 6150                 BSR.S  GETCTRL
000004BA 614E                 BSR.S  GETCTRL
000004BC 0801 0001            BTST   #1,D1
000004C0 6618                 BNE.S  RESETX
000004C2 4CDF 7FFF            MOVEM.L     (SP)+,D0-D7/A0-A6       RESTORE REGISTERS
000004C6 4EF8 FB80            JMP    CTL_RESETV       WATCH IF OUT OF RANGE

000004CA 4879 FF88 TRYMONITOR PEA    MONITOR
              0024
000004D0 4EB8 01A0            JSR    RTN_TO_MONITOR       WILL RETURN IF NO MONITOR
000004D4 584F                 ADDQ   #4,SP                POP MONITOR ADDRESS     (rdq)
000004D6 48E7 FFFE            MOVEM.L D0-D7/A0-A6,-(SP)   SAVE REGISTERS
                        *
              0000 04DA RESETX      EQU *
000004DA 4878 0007            PEA         7                       DEBUGGER(7,0,0)
000004DE 42A7                 CLR.L       -(SP)
000004E0 42A7                 CLR.L       -(SP)
000004E2 2A78 FB7C            MOVEA.L     G_DOLLAR,A5
000004E6 43ED FEEC            LEA         DEBUGGER(A5),A1
000004EA 6100 FE28            BSR         CALLPROC
000004EE 4CDF 7FFF            MOVEM.L     (SP)+,D0-D7/A0-A6       RESTORE REGISTERS
000004F2 4EB8 FB86            JSR         DEBUGESCAPE             GIVE DEBUGGER ANOTHER SHOT

000004F6 4E70                 RESET                               NO DEBUGGER, GIVE UP, CLEAR I/O
000004F8 7010                 MOVEQ       #16,D0                  WAIT 1..2 SECONDS
000004FA 51C9 FFFE LP1        DBRA        D1,LP1
000004FE 51C8 FFFA            DBRA        D0,LP1
00000502 4278 FDC2            CLR         -574                    THEN REBOOT FROM SCRATCH
00000506 4EF8 01C0            JMP         448                     CALL BOOT ROM TO BOOT A SYSTEM
                        *
0000050A 6122         GETCTRL BSR.S  STALL
0000050C 13FC 0005            MOVE.B #5,KBDCOMMAND
              0042 8003
00000514 6118         GA      BSR.S  STALL
00000516 0800 0000            BTST   #0,D0
0000051A 67F8                 BEQ    GA
0000051C 1239 0042            MOVE.B KBDDATA,D1
              8001
00000522 C07C 00F0            AND    #$F0,D0
00000526 B07C 0040            CMP    #$40,D0
0000052A 66E8                 BNE    GA
0000052C 4E75                 RTS
0000052E 1039 0042 STALL      MOVE.B KBDSTATUS,D0
              8003
00000534 0800 0001            BTST   #1,D0
00000538 66F4                 BNE    STALL
```



```
0000053A 4E75                 RTS
                        *                                SUPERCALL CODE MOVED TO ASM_POWERUP     (rdq)
                        *
              0000 053C ASM_INTLEVEL    EQU *                   FUNCTION TO RETURN INTERRUPT LEVEL
0000053C 4E4B                 TRAP    #11                       SHIFT TO SUPERVISOR MODE        (rdq)
0000053E 301F                 MOVE.W  (SP)+,D0                  POP OFF STATUS REGISTER         (rdq)
00000540 46C0                 MOVE    D0,SR                     RETURN TO PREVIOUS MODE         (rdq)
00000542 C0BC 0000            AND.L   #$00000700,D0             EXTRACT LEVEL
              0700
00000548 E048                 LSR     #8,D0                     MOVE IT OVER
0000054A 2F40 0004            MOVE.L  D0,4(SP)                  RETURN INTEGER RESULT
0000054E 4E75                 RTS

              0000 0550 ASM_SETINTLEVEL   EQU *                 PROCEDURE TO SET INTERRUPT LEVEL
00000550 205F                 MOVEA.L (SP)+,A0                  RETURN ADDRESS
00000552 201F                 MOVE.L  (SP)+,D0                  INTEGER PARAMETER
00000554 C07C 0007            AND     #7,D0                     TAKE MOD 8 FOR SAFETY
00000558 E148                 LSL     #8,D0                     MOVE IT OVER
0000055A 4E4B                 TRAP    #11                       MOVE INTO SUPERVISOR MODE
0000055C 321F                 MOVE    (SP)+,D1                  GET CURRENT STATUS REGISTER
0000055E C27C F8FF            AND     #$F8FF,D1                 CLEAR CURRENT LEVEL
00000562 8041                 OR      D1,D0                     COMBINE WITH NEW LEVEL
00000564 46C0                 MOVE    D0,SR                     RESTORE STATUS
00000566 4ED0                 JMP     (A0)

00000568 205F         ASM_IAND MOVEA.L (SP)+,A0
0000056A 201F                 MOVE.L  (SP)+,D0
0000056C C09F                 AND.L   (SP)+,D0
0000056E 2E80                 MOVE.L  D0,(SP)
00000570 4ED0                 JMP     (A0)

00000572 205F         ASM_IOR MOVEA.L (SP)+,A0
00000574 201F                 MOVE.L  (SP)+,D0
00000576 809F                 OR.L    (SP)+,D0
00000578 2E80                 MOVE.L  D0,(SP)
0000057A 4ED0                 JMP     (A0)
```

```
                        ************************  PARITY ERROR CHECKER  ************************
                        *                                                                    JS 9/12/83
                        *     RETURNS WITH CC=EQ IF NOT PARITY ERROR,                        JS 9/12/83
                        *                                                                    JS 9/12/83
0000057C 48E7 C080      P_CHECK MOVEM.L D0-D1/A0,-(SP)    SAVE REGS WE WILL USE            JS 9/12/83
00000580 207C FFFF              MOVEA.L #-4,A0            SETUP FOR VECTOR SWAPPING        JS 9/13/83
         FFFC
00000586 2F10                   MOVE.L  (A0),-(SP)        SAVE OLD BERR VECTOR             JS 9/12/83
00000588 20BC 0000              MOVE.L  #IGNOREBUS,(A0)   SETUP DUMMY BUS ERROR VECTOR     JS 1/23/84
         01A2
0000058E 3F20                   MOVE.W  -(A0),-(SP)                                        JS 9/13/83
00000590 30BC 4EF9              MOVE.W  #JMP,(A0)                                          JS 9/13/83
00000594 7001                   MOVEQ   #1,D0             ASSUME WE HAVE A PARITY ERROR    JS 9/12/83
00000596 487A 002E              PEA     PBERR
0000059A 21CF FB8C              MOVE.L  SP,BESPTEMP
0000059E 3239 005B              MOVE.W  P_STATUS,D1       TRY TO READ STATUS WORD          JS 9/12/83
         0000
000005A4 588F                   ADDQ.L  #4,SP             IF READ OK WE HAVE PARITY ERROR  JS 9/12/83
000005A6 4A40           P_CHECK1 TST    D0                DID WE HAVE A PARITY ERROR?      JS 9/12/83
000005A8 6710                   BEQ.S   P_RTS             IF NOT THEN RESTORE AND RETURN   JS 9/12/83
000005AA 33FC 0000              MOVE.W  #0,P_STATUS       ELSE WE MUST CLEAR THE INTERRUPT JS 9/12/83
         005B 0000
000005B2 33FC 0001              MOVE.W  #1,P_STATUS       AND RE-ENABLE IT                 JS 9/12/83
         005B 0000
000005BA 30DF           P_RTS   MOVE.W  (SP)+,(A0)+       RESTORE OLD BERR VECTOR          JS 9/12/83
000005BC 209F                   MOVE.L  (SP)+,(A0)                                         JS 9/13/83
000005BE 4A40                   TST     D0                SET THE CC                       JS 9/12/83
000005C0 4CDF 0103              MOVEM.L (SP)+,D0-D1/A0    RESTORE REGS                     JS 9/12/83
000005C4 4E75                   RTS                       AND RETURN                       JS 9/12/83
                        *                                                                  JS 9/12/83
000005C6 7000           PBERR   MOVEQ   #0,D0             CLEAR D0 -- NO PARITY ERR        JS 9/12/83
000005C8 4EFA FFDC              JMP     P_CHECK1          AND RETURN FROM BUS ERR ROUTINE  JS 1/23/84
```



```
                        ************************** TIMER JUNK ****************************************

                                def     init_timer      initialize timer: call at powerup/scratch a
                                def     check_timer     call after ~1ms to check w/ bpl
                                def     delay_timer     wait for x microseconds
                                def     asm_ticker      read timer in itcs

                                smode   sysflag2

            0000 0001 timer_present     equ     1               1 if there is a timer, 0 if not

            FFFF FEDA sysflag2          equ     $fffffeda
            FFFF FFFA buserrvec         equ     $fffffffa       bus error vector

                        ********************************************************************************
                        * look_for_timer
                        *   Determine if you have a timer chip on the processor board.
                        *   Try to read from the status register of the chip and see if you get
                        *   a bus error.
                        ********************************************************************************

            0000 05CC look_for_timer equ *
000005CC 2F38 FFFC              move.l  buserrvec+2,-(sp)       save old bus error vector
000005D0 3F38 FFFA              move.w  buserrvec,-(sp)
000005D4 31FC 4EF9              move.w  #$4ef9,buserrvec
         FFFA
000005DA 21FC 0000              move.l  #ignorebus,buserrvec+2  point buserr vector to service routine
         01A2 FFFC
000005E2 487A 0016              pea     buserr_serv
000005E6 21CF FB8C              move.l  sp,besptemp
000005EA 4A39 005F              tst.b   $5f8003                 test for the timer
         8003
000005F0 588F                   addq.l  #4,sp
000005F2 0888 0001              bclr    #timer_present,sysflag2 there is a timer
         FEDA
000005F8 6006                   bra.s   skip1

            0000 05FA buserr_serv equ *
000005FA 08F8 0001              bset    #timer_present,sysflag2 there is no timer
         FEDA

00000600 31DF FFFA skip1    move.w  (sp)+,buserrvec         restore original bus error vector
00000604 21DF FFFC          move.l  (sp)+,buserrvec+2
00000608 4E75               rts

                        ********************************************************************************
                        * init_timer                              initialize the timer
                        *   outputs enabled
                        *   no interrupts
                        *   continuous operating mode
                        *   16 bit for counter 1 and 2, 8 bit for counter 3
                        *   external clock source
                        *   divide by 8 (temporary)
                        *
                        *   CR1         x000000r
```

```
                        *   CR2         x000000a
                        *   CR3         10000101
                        *     x - don't care
                        *     r - counter reset
                        *     a - address bit (selects CR1 or CR3)
                        ********************************************************************************

              0000 060A init_timer equ *
00000608 61C0                   bsr     look_for_timer          look for hardware timer
0000060C 0338 0001              btst    #timer_present,sysflag2 if hardware timer present
         FEDA
00000612 663C                   bne.s   itskp
00000614 41F9 005F              lea     $5f8000,a0              a0 = address of timer
         8000
0000061A 117C 0000              move.b  #$00,3(a0)              address CR3
         0003
00000620 117C 0084              move.b  #$84,1(a0)              set up CR3    ($84 for no divide by 8)
         0001
00000626 117C 0001              move.b  #$01,3(a0)              set up CR2 (address CR1)
         0003
0000062C 117C 0001              move.b  #$01,1(a0)              set up CR1, stop timer
         0001
00000632 117C 00FF              move.b  #$ff,9(a0)              set timer 2 to $ffff
         0009
00000638 117C 00FF              move.b  #$ff,11(a0)
         000B
0000063E 117C 00FF              move.b  #$ff,13(a0)             set timer 3 to $ffff
         000D
00000644 117C 00FF              move.b  #$ff,15(a0)
         000F
0000064A 117C 0000              move.b  #$00,1(a0)              start timer
         0001
00000650 4E75          itskp    rts

                        ********************************************************************************
                        * asm_ticker   function ticker:integer
                        *       sp+4 is the long word to contain the current timer value
                        *
                        ********************************************************************************

              0000 0652 asm_ticker equ *
00000652 42AF 0004              clr.l   4(sp)                   default value
00000656 0838 0001              btst    #timer_present,sysflag2 if hardware timer present
         FEDA
0000065C 661E                   bne.s   rtskp2
0000065E 41F9 005F              lea     $5f8000,a0              address of timer counter 2
         8000
00000664 0148 0009              movep.l 9(a0),d0                read counter 2 and 3
00000668 0348 0009              movep.l 9(a0),d1                read them again
0000066C B181                   eor.l   d0,d1                   see if the upper word has changed
0000066E 4841                   swap    d1
00000670 4A41                   tst.w   d1
00000672 6704                   beq.s   rtskp1
00000674 4841                      swap    d1
00000676 B380                      eor.l   d1,d0                   use second reading if MSW has changed
00000678 2F40 0004 rtskp1  move.l  d0,4(sp)
0000067C 4E75          rtskp2  rts
                        ********************************************************************************
```



```
                        * check_timer                           check the timer
                        *                                   Assumes timer is present!!
                        *
                        *   sp+4 contains a pointer to the timer control data structure:
                        *           timer_control_var: long word
                        *           first_time: boolean  (first byte after timer_control_var)
                        *     if first_time then timer_control_var contains the timeout in milliseconds
                        *     if not first_time then timer_control_var contains timeout clock image
                        *   use bpl or bmi to check for timeout (bpl will branch if not timed out yet)
                        *
                        ********************************************************************************

              0000 067E check_timer equ *

0000067E 48E7 C080              movem.l d0-d1/a0,-(sp)          save registers
                        *                                       read timer
00000682 41F9 005F              lea     $5f8000,a0              address of timer counter 2
         8000
00000688 0148 0009              movep.l 9(a0),d0                read counter 2 and 3
0000068C 0348 0009              movep.l 9(a0),d1                read them again
00000690 B181                   eor.l   d0,d1                   see if the upper word has changed
00000692 4841                   swap    d1
00000694 4A41                   tst.w   d1
00000696 6704                   beq.s   ctskp1
00000698 4841                      swap    d1
0000069A B380                      eor.l   d1,d0                   use second reading if MSW has changed

              0000 069C ctskp1  equ     *                       timer value in d0
0000069C 206F 0010              movea.l 16(sp),a0               a0 = ^timeout value
000006A0 2F6F 000C              move.l  12(sp),16(sp)           move up return address
         0010
000006A6 4A28 0004              tst.b   4(a0)                   first_time?
000006AA 6600 000C              bne     setup_timer             branch if first_time
000006AE B090                      cmp.l   (a0),d0              check for timeout (test with bpl)
000006B0 4CDF 0103                 movem.l (sp)+,d0-d1/a0       restore registers
000006B4 588F                      addq.l  #4,sp                fix up stack
000006B6 4E75                      rts

              0000 06B8 setup_timer equ *
000006B8 2210                   move.l  (a0),d1                 d1 = delay in milliseconds
000006BA ED89                   lsl.l   #6,d1                   multiply by 250
000006BC 9290                   sub.l   (a0),d1
000006BE E389                   lsl.l   #1,d1
000006C0 9290                   sub.l   (a0),d1
000006C2 E389                   lsl.l   #1,d1                   d1 = delay in tics (1 tic = 4 us)
000006C4 9081                   sub.l   d1,d0                   d0 = timeout value
000006C6 20C0                   move.l  d0,(a0)+                store timeout value at a0^
000006C8 4210                   clr.b   (a0)                    first_time = false
000006CA 4CDF 0103              movem.l (sp)+,d0-d1/a0          restore registers
000006CE 588F                   addq.l  #4,sp                   fix up stack
000006D0 4E75                   rts

                        ********************************************************************************
                        * delay_timer                           wait for specified amount of time
                        *   sp+4 contains the amount of time to wait in microseconds (long integer)
                        ********************************************************************************
```

```
                0000 06D2 delay_timer  equ  *
000006D2 48E7 C080          movem.l d0-d1/a0,-(sp)
000006D6 0838 0001          btst    #timer_present,sysflag2 check for hardware timer
         FEDA

                        * Use the timer to measure delay.
                        * There is 218 cycles of overhead in addition to the timer amount.
                        * It takes 351 cycles to go through the routine once.

000006DC 6634               bne.s   dlskp2
000006DE 41F9 005F          lea     $5f8000,a0              address of timer
         8000
000006E4 0148 0009          movep.l 9(a0),d0                read counter 2 and 3
000006E8 0348 0009          movep.l 9(a0),d1                read them again
000006EC B181               eor.l   d0,d1                   see if the upper word has changed
000006EE 4841               swap    d1
000006F0 4A41               tst.w   d1
000006F2 6704               beq.s   dlskp1
000006F4 4841               swap    d1
000006F6 B380               eor.l   d1,d0                   use second reading if MSW has changed
000006F8 222F 0010 dlskp1   move.l  16(sp),d1               d1 = delay in microseconds
000006FC E489               lsr.l   #2,d1                   d1 = delay in tics (1 tic = 4 us)
000006FE 5781               subq.l  #3,d1                   subtract overhead outside timing loop
00000700 9081               sub.l   d1,d0                   d0 = timeout value

00000702 0348 0009 dlloop1  movep.l 9(a0),d1                read counter 2 and 3
00000706 B280               cmp.l   d0,d1                   compare timer value to timeout value
00000708 6AF8               bpl.s   dlloop1
0000070A 4CDF 0103          movem.l (sp)+,d0-d1/a0
0000070E 2E9F               move.l  (sp)+,(sp)
00000710 4E75               rts

                        * Use timing loop if no timer is present.
                        * Number of cycles = 230+31n

00000712 222F 0010 dlskp2   move.l  16(sp),d1               d1 = delay in microseconds
00000716 E489               lsr.l   #2,d1                   d1 = number of loops (1 loop = 4 us)
00000718 5F81               subq.l  #7,d1                   subtract overhead outside loop (230/31)
0000071A 5381               subq.l  #1,d1                   timing loop
0000071C 4E71               nop
0000071E 4E71               nop
00000720 6AF8               bpl.s   dlloop2
00000722 4CDF 0103          movem.l (sp)+,d0-d1/a0
00000726 2E9F               move.l  (sp)+,(sp)              pop parameter
00000728 4E75               rts
```



```
                        *       TRAPS:
                        *
                        *       ADDRESS         TRAP    ESCAPECODE      USUAL MEANING
                        *
                        *       $FFFFFF3A       15      (NONE)          DEBUGGER
                        *       $FFFFFF40       14      -21             UNASSIGNED
                        *       $FFFFFF46       13      -21             UNASSIGNED
                        *       $FFFFFF4C       12      -21             UNASSIGNED
                        *       $FFFFFF52       11      (NONE)          SUPERVISOR MODE
                        *       $FFFFFF58       10      "N"             ESCAPE N
                        *       $FFFFFF5E       9       (NONE)          NON LOCAL GOTO
                        *       $FFFFFF64       8        -3             DEREFERENCE NIL POINTER
                        *       $FFFFFF6A       7        -8             VALUE RANGE ERROR
                        *       $FFFFFF70       6        -9             CASE STATEMENT ERROR
                        *       $FFFFFF76       5        -5             DIVIDE BY ZERO
                        *       $FFFFFF7C       4        -4             INTEGER OVERFLOW
                        *       $FFFFFF82       3       -10             IORESULT <> 0
                        *       $FFFFFF88       2        -2             STACK OVERFLOW
                        *       $FFFFFF8E       1       (-2 IF ANY)     LINK A6 EMULATOR WITH STACK CHECK
                        *       $FFFFFF94       0       (NONE)          PASCAL LINE BREAKPOINT

                        *       SYSTEM EXCEPTIONS:
                        *
                        *       ADDRESS                 ESCAPECODE      USUAL MEANING
                        *
                        *       $FFFFFFC4               -13             1111 OPCODE
                        *       $FFFFFFC8               -13             1010 OPCODE
                        *       $FFFFFFD0               (NONE)          TRACE
                        *       $FFFFFFD6               -14             PRIVILEGE VIOLATION
                        *       $FFFFFFDC                -4             INTEGER OVERFLOW (TRAPV)
                        *       $FFFFFFE2                -8             CHK INSTRUCTION
                        *       $FFFFFFE8                -5             DIVIDE BY ZERO (HARDWARE)
                        *       $FFFFFFEE               -13             ILLEGAL INSTRUCTION
                        *       $FFFFFFF4               -11             ADDRESS ERROR
                        *       $FFFFFFFA               -12             BUS ERROR

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

RANDOM

Description

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.

```
PAGE   1 [3.0] 12/26/84 22:50:13  ASSEMBLY OF RANDOM.TEXT                                   *** File name: RANDOM ***

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


PAGE   2 [3.0] 12/26/84 22:50:13  ASSEMBLY OF RANDOM.TEXT                                   *** File name: RANDOM ***

   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
DFC	STREG	0		00000008
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
SFC	STREG	0		00000009
SP	AREG	0		00000007
SR	STREG	0		00000006
USP	STREG	0		00000007
VBR	STREG	0		0000000A

Description

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

```
PAGE   1 [3.0] 12/26/84 20:59:59  ASSEMBLY OF ROMCALL.TEXT                                  *** File name: ROMCALL ***

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,asm_cache_on,asm_cache_off
    5                                   def     boot_lifhead,boot_findfile
    6                                   def     boot_minit,boot_mfopen,boot_mfclose,boot_mread
    7                                   refa    sysglobals,msysflags
    8
    9               FFFF FFFE escapecode        equ sysglobals-2
   10               FFFF FFF6 recoverblock      equ sysglobals-10
   11               FFFF FFB8 intvec2           equ -72                 vector for interrupt level 2
   12               0000 3FFE boot_id           equ $3FFE
   13
   14                         *  hard coded addresses:
   15               0000 0120 flpyread          equ $120        288
   16               0000 0124 flpy_wrt          equ $124        292
   17               0000 0128 fintrupt          equ $128        296
   18               0000 012C flpyinit          equ $12C        300
   19               0000 0130 flpymread         equ $130        304
   20               0000 0134 flpymwrite        equ $134        308
   21               0000 0144 f_pwr_on          equ $144        324
   22               0000 018C lifhead           equ     $18C
   23                         *findfile          equ     $190
   24
   25               0000 4004 minit             equ     $4004
   26               0000 4008 mfopen            equ     $4008
   27               0000 4010 mfclose           equ     $4010
   28               0000 400C mread             equ     $400C
   29
   30               0000 0000 A         equ     a0      address of boot ROM routine
   31               0000 0004 R         equ     a4      return address
   32               0000 0004 F         equ     d4      function result
   33
   34
   35               0000 0000 boot_mfopen       equ  *
   36    00000000 41F8 4008           lea     mfopen,A
   37
   38    00000004 285F        f2_16   movea.l (sp)+,R                         get return address
   39    00000006 4CDF 001E           movem.l (sp)+,d1-d4                     get parameters
   40    0000000A 2F0C                move.l  R,-(sp)                         put return address back
   41    0000000C 613C                bsr.s   fixit                           get into boot ROM environment
   42    0000000E 4267                clr.w   -(sp)                           function result
   43    00000010 48E7 7800           movem.l d1-d4,-(sp)                     push parameters
   44    00000014 6010                bra.s   rom_fun
   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 381F                move.w  (sp)+,F                         get function result
```

```
PAGE   2 [3.0] 12/26/84 20:59:59  ASSEMBLY OF ROMCALL.TEXT                                  *** File name: ROMCALL ***

   59    0000002A 614E                bsr.s   unfix                           return to user environment
   60    0000002C 3F44 0004           move.w  F,4(sp)                         put back function result
   61    00000030 4E75                rts                                     return
   62
   63               0000 0032 asm_flpyinit equ *
   64    00000032 41F8 012C           lea     flpyinit,A
   65    00000036 285F                movea.l (sp)+,R                         get return address
   66    00000038 4C9F 000E           movem.w (sp)+,d1-d3                     get parameters
   67    0000003C 2F0C                move.l  R,-(sp)                         put return address back
   68    0000003E 610A                BSR.S   FIXIT                   (rdq)   get into boot ROM environment
   69    00000040 48A7 7000           movem.w d1-d3,-(sp)                     push parameters
   70
   71    00000044 4E90        rom_jsr jsr     (A)                             call boot ROM
   72    00000046 6132        ROM_XIT bsr.s   unfix                           restore user envirnoment
   73    00000048 4E75                rts                                     return to user
   74
   75    0000004A 43FA 00AE fixit   LEA     ROM_RECOVER,A1                  (rdq)
   76    0000004E 285F                movea.l (sp)+,R                         fetch return address
   77    00000050 4E4B                trap    #11                             get into supervisor mode
   78    00000052 47F8 FFB8           lea     intvec2,a3                      save floppy vector
   79    00000056 3F1B                move.w  (a3)+,-(sp)                     save jmp/jsr
   80    00000058 2F1B                move.l  (a3)+,-(sp)                     save address
   81    0000005A 273C 0000           move.l  #fintrupt,-(a3)                 set up boot ROM floppy vector
              0128
   82    00000060 373C 4EF9           move.w  #$4EF9,-(a3)                    (jmp)
   83    00000064 4E55 FFFE           link    a5,#-2                          primitive try/recover
   84    00000068 4851                PEA     (A1)                            (rdq) RECOVER ADDRESS
   85    0000006A 4857                pea     (sp)
   86    0000006C 0C78 0003           CMPI.W  #3,BOOT_ID                      3.0 BUG FIX jws
              3FFE
   87    00000072 6604                BNE.S   FIXIT1                          3.0 BUG FIX jws
   88    00000074 6100 00E6           BSR     ASM_CACHE_OFF                   3.0 BUG FIX jws
   89    00000078 4ED4        FIXIT1  jmp     (R)                             return to call
   90
   91    0000007A 0C78 0003 UNFIX   CMPI.W  #3,BOOT_ID                      3.0 BUG FIX jws
              3FFE
   92    00000080 6604                BNE.S   UNFIX1                          3.0 BUG FIX jws
   93    00000082 6100 00EA           BSR     ASM_CACHE_ON                    3.0 BUG FIX jws
   94    00000086 285F        UNFIX1  movea.l (sp)+,R                         fetch return address
   95    00000088 4E5D                unlk    a5                              pop recover stuff
   96    0000008A 47F8 FFBE           lea     intvec2+6,a3                    restore floppy vector
   97    0000008E 271F                move.l  (sp)+,-(a3)                     address
   98    00000090 371F                move.w  (sp)+,-(a3)                     jmp
   99    00000092 46DF                move    (sp)+,sr                        restore user mode
  100    00000094 4ED4                jmp     (R)                             return to call
  101
  102
  103
  104               0000 0096 boot_mread       equ  *
  105    00000096 41F8 400C           lea     mread,A
  106
  107    0000009A 285F        byte_14 movea.l (sp)+,R                         get return address
  108    0000009C 4C9F 00FE           movem.w (sp)+,d1-d7                     get parameters
  109    000000A0 2F0C                move.l  R,-(sp)                         put return address back
  110    000000A2 61A6                bsr.s   fixit                           get into boot ROM environment
  111    000000A4 48A7 7F00           movem.w d1-d7,-(sp)                     push parameters
  112    000000A8 609A                bra.s   rom_jsr
```

```
  113
  114
  115
  116                0000 00AA boot_mfclose    equ *
  117     000000AA 41F8 4010          lea     mfclose,A
  118     000000AE 6004               bra.s   no_parm
  119
  120                0000 00B0 asm_f_pwr_on equ *
  121     000000B0 41F8 0144          lea     f_pwr_on,A                  boot ROM mini reset
  122
  123     000000B4 6194       no_parm bsr.s   fixit                       get into boot ROM environment
  124     000000B6 608C               bra.s   rom_jsr
  125
  126
  127
  128                0000 00B8 boot_lifhead equ *
  129     000000B8 41F8 018C          lea     lifhead,A
  130     000000BC 600A               bra.s   byte_8
  131
  132                0000 00BE asm_flpy_wrt equ *
  133     000000BE 41F8 0124          lea     flpy_wrt,A                  boot ROM single-sector write
  134     000000C2 6004               bra.s   byte_8
  135
  136                0000 00C4 asm_flpyread equ *
  137     000000C4 41F8 0120          lea     flpyread,A                  boot ROM single-sector read
  138
  139     000000C8 285F       byte_8  movea.l (sp)+,R                     get return address
  140     000000CA 4CDF 0006          movem.l (sp)+,d1-d2                 get parameters
  141     000000CE 2F0C               move.l  R,-(sp)                     put return address back
  142     000000D0 6100 FF78          bsr     fixit                       get into boot ROM environment
  143     000000D4 48E7 6000          movem.l d1-d2,-(sp)                 push parameters
  144     000000D8 6000 FF8A          bra     rom_jsr
  145
  146
  147
  148                0000 00DC asm_flpymwrite equ *
  149     000000DC 41F8 0134          lea     flpymwrite,A                boot ROM multisector write
  150     000000E0 6004               bra.s   byte_12
  151
  152                0000 00E2 asm_flpymread equ *
  153     000000E2 41F8 0130          lea     flpymread,A                 boot ROM multisector read
  154
  155     000000E6 285F       byte_12 movea.l (sp)+,R                     get return address
  156     000000E8 4CDF 000E          movem.l (sp)+,d1-d3                 get parameters
  157     000000EC 2F0C               move.l  R,-(sp)                     put return address back
  158     000000EE 6100 FF5A          bsr     fixit                       get into boot ROM environment
  159     000000F2 48E7 7000          movem.l d1-d3,-(sp)                 push parameters
  160     000000F6 6000 FF4C          bra     rom_jsr
  161
  162                0000 00FA rom_recover equ *                         escape from boot ROM
  163     000000FA 381F               move.w  (sp)+,d4                    get escapecode
  164     000000FC 6100 FF7C          bsr     unfix                       return to user environment
  165     00000100 3B44 FFFE          move.w  d4,escapecode(a5)           restore escapecode
  166     00000104 4E4A               trap    #10                         escape(escapecode)
  167
  168
  169
```



```
  170                FFFF FDD2 FUBUFFER EQU     $FFFFFDD2       FLOPPY UNIVERSAL BUFFER (RQ)
  171
  172                0000 0106 BOOT_FINDFILE   EQU *
  173     000001C6 225F               MOVEA.L (SP)+,A1            RETURN ADDRESS
  174     000001C8 241F               MOVE.L  (SP)+,D2            GET DIR START
  175     000001CA 261F               MOVE.L  (SP)+,D3            GET DIR LEN
  176     000001CC 205F               MOVEA.L (SP)+,A0            GET FILE NAME ADDR
  177     000001CE 285F               MOVEA.L (SP)+,A4            GET ADDR OF CAT ENTRY
  178     00000110 2F09               MOVE.L  A1,-(SP)            RESTORE RETURN ADDRESS
  179     00000112 45F8 FDD2 NXTDSECTOR LEA   FUBUFFER,A2
  180     00000116 48E7 30A8          MOVEM.L D2/D3/A0/A2/A4,-(SP)    SAVE REGS
  181     0000011A 2F02               MOVE.L  D2,-(SP)
  182     0000011C 2F0A               MOVE.L  A2,-(SP)            READ THE NEXT
  183     0000011E 61A4               BSR.S   ASM_FLPYREAD        DIRECTORY SECTOR
  184     00000120 4CDF 150C          MOVEM.L (SP)+,D2/D3/A0/A2/A4  RESTORE REGS
  185     00000124 7208               MOVEQ   #8,D1               (8 ENTRIES/SECTOR)
  186     00000126 0C6A FFFF CHECKTYPE CMPI.W #-1,10(A2)          CHECK LOGICAL EOD
                   000A
  187     0000012C 6726               BEQ.S   NOTFOUND
  188     0000012E 4A6A 000A          TST.W   10(A2)              IGNORE PURGED FILES
  189     00000132 6712               BEQ.S   NXTDENTRY
  190     00000134 2248               MOVEA.L A0,A1               SAVE NAME POINTER
  191     00000136 264A               MOVEA.L  A2,A3              COPY ENTRY POINTER
  192     00000138 7804               MOVEQ    #4,D4              SET CHARACTER COUNTER
  193     0000013A B749       CHKNAME CMPM.W  (A1)+,(A3)+         MATCH CHARACTERS
  194     0000013C 56CC FFFC          DBNE    D4,CHKNAME
  195     00000140 6604               BNE.S   NXTDENTRY
  196     00000142 288A               MOVE.L  A2,(A4)             STORE THE CAT ENTRY ADDR.
  197     00000144 4E75               RTS
  198
  199     00000146 D4FC 0020 NXTDENTRY ADDA   #32,A2              INCREMENT
  200     0000014A 5381               SUBQ.L  #1,D1               TO NEXT ENTRY
  201     0000014C 6ED8               BGT.S   CHECKTYPE
  202     0000014E 5282               ADDQ.L  #1,D2               INCREMENT
  203     00000150 5383               SUBQ.L  #1,D3               TO NEXT DIRECTORY
  204     00000152 6EBE               BGT.S   NXTDSECTOR          SECTOR
  205     00000154 3B7C FFFF NOTFOUND MOVE.W  #-1,ESCAPECODE(A5)  ESCAPECODE := -1
                   FFFE
  206     0000015A 4E4A               TRAP    #10
  207
  208                         *---rdg---ROUTINES TO TURN CACHE OFF AND ON--------------------------------
  209                005F 400E CACHE_CTL EQU   $5F400E
  210
  211                0000 015C ASM_CACHE_OFF   EQU *
  212     0000015C 0838 0000          BTST    #0,MSYSFLAGS
                   0000
  213     00000162 6708               BEQ.S   XIT
  214     00000164 0279 0003          ANDI.W  #$0003,CACHE_CTL        CLEAR TST, ERR AND CACHE ENABLE BITS
                   005F 400E
  215     0000016C 4E75       XIT     RTS
  216                0000 016E ASM_CACHE_ON    EQU *
  217     0000016E 0838 0000          BTST    #0,MSYSFLAGS
                   0000
  218     00000174 67F6               BEQ     XIT
  219     00000176 91C8               SUBA.L  A0,A0           READ 16K BYTES FROM HIGH MEM DOWN
  220     00000178 303C 0FFF          MOVE.W  #4095,DC        TO FLUSH THE CACHE
  221     0000017C 2220       CACHE01 MOVE.L  -(A0),D1
```

```
  222    0000017E 51C8 FFFC          DBRA    D0,CACHE01
  223    00000182 41F9 005F          LEA     CACHE_CTL,A0
                  400E
  224    00000188 3010               MOVE.W  (A0),D0                 FETCH CURRENT STATUS
  225    0000018A 0240 0007          ANDI.W  #$0007,D0               CLEAR TST AND ERR BITS
  226    0000018E 08C0 0002          BSET    #2,D0                   SET CACHE ENABLE BIT
  227    00000192 3080               MOVE.W  D0,(A0)                 PUT IT BACK
  228    00000194 4E75               RTS
  229                        *---rdq-------------------------------------------------------------------
  230
  231                                nosyms
  232                                end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

Description

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

Usage

RS is used for the 98626 interface and the Model 216/217 computers' built-in RS-232 interface.

Requirements

RS_DRV and COMASM.

```
PASS 1 COMPLETE. ERRORS: 0
    2                       ********************************************************************************
    3                       *
    4                       *       COPYRIGHT (C) 1984 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  - 5-3-84 BY ---------
   25                       *       Release - 3.0
   26                       *
   27                       *
   28                       *       Source  - RS: RS.TEXT
   29                       *       Object  - RS: RS.CODE
   30                       *
   31                       *
   32                       *
   33                       *
   34                       ********************************************************************************
   35                       *
   36                       *
   37                       *           RELEASED        VERSION 3.0
   38                       *
   39                       *
   40                       ********************************************************************************
   41                       *
   42                       *       CHANGES (since 2.0):
   43                       *           {aaa} -- changes for ignore parity error register 20.
   44                       *
   45                       *           {ttt} -- timing changes for 680xx -- JS 8/3/83, 5/3/84
   46                       *
   47                       ********************************************************************************
```



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

```
                        ********************************************************************************
                        *
                        *       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
                                REFA    DELAY_TIMER,CHECK_TIMER             ttt JS 8/3/83
                                LMODE   DELAY_TIMER,CHECK_TIMER             ttt JS 8/3/83


                                INCLUDE IOLIB:COMDCL
```



```
                        ********************************************************************************
                        *
                        *       modified:  02/22/82  JPC    added parm to user EOT & ISR proc's
                        *                  08/01/83  JS     added timer_present and sysflag2 equ'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

                0000 0001 TIMER_PRESENT  EQU   1        JS  8/1/83  SYSFLAG2_BIT -- 0=>TIMER PRESENT
                FFFF FEDA SYSFLAG2       EQU   $FFFFFEDA          JS  8/1/83
```



```
                        *
                        *       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   0   -   -   -   -   -   -
                        *       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
```

```
349                     *
350                     *       ROUTINE USAGE SUMMARY
351                     *
352                     *       ROUTINE         CALLS
353                     *       -------         -----
354                     *       queue_space     <none>
355                     *       queue_empty     <none>
356                     *       queue_full      <none>
357                     *       inqueue         <none>
358                     *       outqueue        <none>
359                     *       init_queue      <none>
360                     *       check_queue     <none>
361                     *       check_dsr_cts   <none>
362                     *       wait            check_queue, check_dsr_cts (both indirectly)
363                     *       send            wait, check_dsr_cts
364                     *       get_char        outqueue, queue_space, send
365                     *       wait_send       send, check_error
366                     *       wait_get        wait, check_error, get_char, check_queue
367                     *       check_error     ioescape
368                     *       soft_reset      init_queue, ABORT_IO
369                     *       connect         soft_reset
370                     *       disconnect      <none>
371                     *       rdivu           <none>
372                     *       check_xfer_in   ioescape
373                     *       check_xfer_out  ioescape
374                     *       clear_xfer      <none>
375                     *       set_xfer        <none>
376                     *       dump_buffer     queue_empty, get_char, clear_xfer, LOGEOT
377                     *       ioescape        <none>
378                     *       init            init_queue, ABORT_IO
379                     *       rdb             connect, check_error, check_xfer_in, wait_get
380                     *       wtb             connect, check_error, check_xfer_out, wait_send
381                     *       rdw             connect, check_error, check_xfer_in, wait_get
382                     *       wtw             connect, check_error, check_xfer_out, wait_send
383                     *       rds             queue_empty, get_char, check_error, rdivu, ioescape,
384                     *                       connect
385                     *       wtc             check_error, soft_reset, connect, disconnect, rdivu,
386                     *                       ioescape
387                     *       isr             queue_space, queue_full, inqueue, check_dsr_cts,
388                     *                       send, dump_buffer, LOGEOT
389                     *       tfr             connect, check_error, dump_buffer, set_xfer
390
```



```
392                     *
393                     *
394                     *       ROUTINE         CALLED BY
395                     *       -------         ---------
396                     *       queue_space     get_char, isr
397                     *       queue_empty     dump_buffer, rds (6, 10)
398                     *       queue_full      isr
399                     *       inqueue         isr
400                     *       outqueue        get_char
401                     *       init_queue      init, soft_reset
402                     *       check_queue     wait_get (with wait)
403                     *       check_dsr_cts   isr, send (with and without wait)
404                     *       wait            wait_get, wait_send, send
405                     *       send            isr, wait_send, get_char
406                     *       get_char        rds(6), dump_buffer, wait_get
407                     *       wait_send       wtb, wtw
408                     *       wait_get        rdb, rdw,
409                     *       check_error     rdb, wtb, rdw, wtw, rds(6), wtc(6), tfr,
410                     *                       wait_send, wait_get
411                     *       soft_reset      wtc(14), connect
412                     *       connect         rdb, wtb, rdw, wtw, wtc(12), tfr
413                     *       disconnect      wtc(12)
414                     *       rdivu           rds(3), wtc(3)
415                     *       check_xfer_in   rdb, rdw,
416                     *       check_xfer_out  wtb, wtw
417                     *       clear_xfer      isr, dump_buffer
418                     *       set_xfer        tfr
419                     *       dump_buffer     tfr, isr
420                     *       ioescape        rds, wtc, tfr, check_error, check_xfer_in,
421                     *                       check_xfer_out
422                     *       ABORT_IO        init, soft_reset
423                     *       LOGEOT          isr, dump_buffer
424
425
```

```
                        ********************************************************************************
                        *
                        *       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 DIVO            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 IGNORE_PE       EQU     CONV_CHAR+1     1  (aaa)
                      *empty                                  1  (aaa)
            0000 0048 Q_DESCRIPT      EQU     IGNORE_PE+2        (aaa)
            0000 0048 Q_SIZE          EQU     Q_DESCRIPT      2
            0000 004A Q_IN            EQU     Q_DESCRIPT+2    2
            0000 004C Q_OUT           EQU     Q_DESCRIPT+4    2
            0000 004E Q_BUFFER        EQU     Q_DESCRIPT+6    134 the rest is internal buffer (aaa)
                        *                                       ---
                        *                                       160  total space used


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

            0000 00A0 TEMP_SIZE       EQU     160             Pascal declaration of size of misc. space
            0000 0086 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 005E XON_SIZE        EQU     BUFFER_SIZE-XOFF_SIZE
            0000 0014 REG_MAX         EQU     20              maximum control/status register number (aaa)
```

```
                        ********************************************************************************
                        *
                        *       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 7046               MOVEQ   #70,D0          *  wait AT LEAST 25 us
                        *
                        * Count changed from 32 to 70 to allow for 16MHz processors JS 8/3/83
                        *
    00000018 51C8 FFFE          DBRA    D0,*            *  ( not yet timed -- #32 was initial est.)

                        *
                        *       global attribute initialization
                        *
                        *                                   -- init queue descriptors --
    0000001C 6100 0A42          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)
    0000003C 422A 0046          CLR.B   IGNORE_PE(A2)           {aaa}
                        *                                   -- set flags --
    00000040 157C 0001          MOVE.B  #1,RECEIVING(A2)
                003C
    00000046 157C 0001          MOVE.B  #1,XMITTING(A2)
                003D
                        *                                   -- default characters --
```



```
    0000004C 157C 0011          MOVE.B  #DC1,XON_CHAR(A2)
                0041
    00000052 157C 0013          MOVE.B  #DC3,XOFF_CHAR(A2)
                0042
    00000058 157C 0005          MOVE.B  #ENQ,ENQ_CHAR(A2)
                0043
    0000005E 157C 0006          MOVE.B  #ACK,ACK_CHAR(A2)
                0044
    00000064 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 --
    0000006A 700A               MOVEQ   #10,D0          * SET UP FOR 2400 BAUD (98644 DEFAULT)
    0000006C 1C29 0001          MOVE.B  ID_REG(A1),D6   * GET ID REG
    00000070 0886 0007          BCLR    #7,D6           * GET RID OF REMOTE BIT
    00000074 BC3C 0002          CMP.B   #2,D6           * IS THIS A 98626 ?
    00000078 6604               BNE.S   INITALL         * IF NOT THEN USE DEFAULT
    0000007A 1029 0005          MOVE.B  BAUD_SW(A1),D0  * IF 98626 THEN   read switches
    0000007E E308       INITALL LSL.B   #1,D0           *   multiply by two to get to words
    00000080 0200 001E          ANDI.B  #$1E,D0         *   mask
    00000084 4880               EXT     D0              *
                        *
    00000086 41FA 0034          LEA     BAUD,A0         *   a0 :=  pointer to baud table
    0000008A D0C0               ADDA    D0,A0           *              entry
                        *
    0000008C 08E9 0007          BSET    #7,LINE_CONT(A1) *   set DLAB to get to divisor latches
                0017
    00000092 1358 0013          MOVE.B  (A0)+,DIV1(A1)  *
    00000096 1350 0011          MOVE.B  (A0),DIV0(A1)   *


                        *                                   -- SET LINE CHARACTERISTICS --
    0000009A 7003               MOVEQ   #3,D0           * SET DEFAULTS FOR 98644
    0000009C BC3C 0002          CMP.B   #2,D6           * CHECK IF 98626
    000000A0 6604               BNE.S   SETALL          * IF NOT USE DEFAULTS
    000000A2 1029 0007          MOVE.B  LINE_SW(A1),D0  *  get handshake and line status defaults
    000000A6 1200       SETALL  MOVE.B  D0,D1           *
    000000A8 0200 003F          ANDI.B  #$3F,D0         *  mask and set default line status
    000000AC 1340 0017          MOVE.B  D0,LINE_CONT(A1) *     (also restores dlab)
                        *                                   -- SAVE HANDSHAKE --
    000000B0 0201 00C0          ANDI.B  #$C0,D1         *  handshake is top two bits
    000000B4 EA09               LSR.B   #5,D1           *  adjust it (for jump tables)
    000000B6 1541 0040          MOVE.B  D1,S_HANDSH(A2) *  save it

    000000BA 4E75               RTS
                        *
                        *       baud rate table (for switches)
                        *
    000000BC 0C00       BAUD    DC.W    3072            *    50 Baud
    000000BE 0800               DC.W    2048            *    75
    000000C0 0574               DC.W    1396            *   110
    000000C2 0476               DC.W    1142            *   134.5
    000000C4 0400               DC.W    1024            *   150
```

```
000000C6 0300              DC.W    768          *   200
000000C8 0200              DC.W    512          *   300
000000CA 0100              DC.W    256          *   600
000000CC 0080              DC.W    128          *  1200
000000CE 0055              DC.W     85          *  1800
000000D0 0040              DC.W     64          *  2400
000000D2 002B              DC.W     43          *  3600
000000D4 0020              DC.W     32          *  4800
000000D6 0015              DC.W     21          *  7200
000000D8 0010              DC.W     16          *  9600
000000DA 0008              DC.W      8          * 19200
```



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

          0000 00DC RS_RS_RDB       EQU *

                    *
                    *   Pascal interface overhead
                    *
000000DC 205F               MOVEA.L (SP)+,A0            get return address
000000DE 265F               MOVEA.L (SP)+,A3            get VAR address
000000E0 245F               MOVEA.L (SP)+,A2            get temp address
000000E2 226A 0020          MOVEA.L C_ADR(A2),A1        get card address
000000E6 4850               PEA     (A0)                restore return address
                    *
                    *   Card overhead (any of the three can do an ioescape)
                    *
000000E8 6100 0770          BSR     CONNECT             make sure the card is active
000000EC 6100 07EA          BSR     CHECK_ERROR         check for errors saved by ISRs
000000F0 6100 0726          BSR     CHECK_XFER_IN       make sure no input transfers active

000000F4 6100 0810          BSR     WAIT_GET            get character with wait
000000F8 1682               MOVE.B  D2,(A3)             return the character
000000FA 4E75               RTS
```

```
PAGE  17 [3.0] 12/26/84 21:34:08 RS232 DRIVERS -- WRITE BYTE                    *** File name: RS ***

 667                    ********************************************************************************
 668                    *
 669                    *       write byte
 670                    *
 671                    ********************************************************************************
 672
 673            0000 00FC RS_RS_WTB       EQU *
 674
 675                    *
 676                    *   Pascal interface overhead
 677                    *
 678    000000FC 205F           MOVEA.L (SP)+,A0              get return address
 679    000000FE 161F           MOVE.B  (SP)+,D3              get char to be written
 680    00000100 245F           MOVEA.L (SP)+,A2              get temp address
 681    00000102 226A 0020      MOVEA.L C_ADR(A2),A1          get card address
 682    00000106 4850           PEA     (A0)                  restore return address
 683                    *
 684                    *   Card overhead
 685                    *
 686    00000108 6100 0750      BSR     CONNECT               autoconnect
 687    0000010C 6100 07CA      BSR     CHECK_ERROR           check for errors found by ISRs
 688    00000110 6100 070E      BSR     CHECK_XFER_OUT        make sure no output transfers are active
 689
 690    00000114 6100 07E2      BSR     WAIT_SEND             send the character
 691    00000118 4E75           RTS
 692
```

```
PAGE  18 [3.0] 12/26/84 21:34:08 RS232 DRIVERS -- READ WORD                     *** File name: RS ***

 695                    ********************************************************************************
 696                    *
 697                    *       read word
 698                    *
 699                    ********************************************************************************
 700
 701            0000 011A RS_RS_RDW       EQU *
 702
 703    0000011A 205F           MOVEA.L (SP)+,A0              get return address
 704    0000011C 265F           MOVEA.L (SP)+,A3              get VAR address
 705    0000011E 245F           MOVEA.L (SP)+,A2              get temp address
 706    00000120 226A 0020      MOVEA.L C_ADR(A2),A1          get card address
 707    00000124 4850           PEA     (A0)                  restore return address
 708                    *
 709                    *   Card overhead (any of the three can do an ioescape)
 710                    *
 711    00000126 6100 0732      BSR     CONNECT               make sure the card is active
 712    0000012A 6100 07AC      BSR     CHECK_ERROR           check for errors saved by ISRs
 713    0000012E 6100 06E8      BSR     CHECK_XFER_IN         make sure no input transfers are active
 714
 715    00000132 6100 07D2      BSR     WAIT_GET              get character with wait
 716    00000136 E14A           LSL.W   #8,D2                 shift first character
 717    00000138 6100 07CC      BSR     WAIT_GET              get second character
 718    0000013C 3682           MOVE.W  D2,(A3)               return the word
 719    0000013E 4E75           RTS
```

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

              0000 0140 RS_RS_WTW       EQU *

00000140 205F                 MOVEA.L (SP)+,A0                get return address
00000142 361F                 MOVE.W  (SP)+,D3                get word to be written
00000144 245F                 MOVEA.L (SP)+,A2                get temp address
00000146 226A 0020            MOVEA.L C_ADR(A2),A1            get card address
0000014A 4850                 PEA     (A0)                    restore return address
                        *
                        *   Card overhead (any of the three can do an ioescape)
                        *
0000014C 6100 070C            BSR     CONNECT                 make sure the card is active
00000150 6100 0786            BSR     CHECK_ERROR             check for errors saved by ISRs
00000154 6100 06CA            BSR     CHECK_XFER_OUT          make sure no output transfers are active

00000158 E05B                 ROR.W   #8,D3                   position the first character
0000015A 6100 079C            BSR     WAIT_SEND               send it
0000015E E05B                 ROR.W   #8,D3                   position the second character
00000160 6100 0796            BSR     WAIT_SEND               send it

00000164 4E75                 RTS
```



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

              0000 0166 RS_RS_RDS       EQU *
                        *
                        *   Pascal Interface Overhead
                        *

00000166 205F                 MOVEA.L (SP)+,A0                get return address
00000168 265F                 MOVEA.L (SP)+,A3                get VAR address
0000016A 321F                 MOVE.W  (SP)+,D1                get register number
0000016C 245F                 MOVEA.L (SP)+,A2                get temp address
0000016E 226A 0020            MOVEA.L C_ADR(A2),A1            get card address
00000172 4850                 PEA     (A0)                    restore return address

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

00000174 4A41                 TST.W   D1                      check for negative register number
00000176 6D3E                 BLT.S   STS_ERROR               if so goto common ioescape routine
00000178 B27C 0014            CMP.W   #REG_MAX,D1             check for too large register number
0000017C 6E38                 BGT.S   STS_ERROR

0000017E 4253                 CLR.W   (A3)                    clear the top half of the return word
00000180 528B                 ADDQ.L  #1,A3                   point a3 to lower half byte of return word

00000182 E349                 LSL.W   #1,D1                   get ready for (word) table jump
00000184 323B 1006            MOVE.W  STS_TABLE(D1),D1        get offset from status table
00000188 4EFB 1002            JMP     STS_TABLE(D1)           do the indexed jump

              0000 018C STS_TABLE       EQU *
0000018C 0030                 DC.W    STS_0-STS_TABLE         ID register
0000018E 0036                 DC.W    STS_1-STS_TABLE         Interrupt status register
00000190 003C                 DC.W    STS_2-STS_TABLE         Busy bits register
00000192 006E                 DC.W    STS_3-STS_TABLE         Baud rate
00000194 00D6                 DC.W    STS_4-STS_TABLE         Character control register
00000196 00EA                 DC.W    STS_5-STS_TABLE         Modem control register
00000198 00F0                 DC.W    STS_6-STS_TABLE         Data in register
0000019A 0114                 DC.W    STS_7-STS_TABLE         Optional circuits register
0000019C 012E                 DC.W    STS_8-STS_TABLE         Interrupt Enable Mask register
0000019E 0134                 DC.W    STS_9-STS_TABLE         Interrupt Cause register
000001A0 013A                 DC.W    STS_10-STS_TABLE        UART Status register
000001A2 0164                 DC.W    STS_11-STS_TABLE        Modem Status register
000001A4 0184                 DC.W    STS_12-STS_TABLE        Connect/Disconnect register
000001A6 018A                 DC.W    STS_13-STS_TABLE        Hardware handshake register
000001A8 0190                 DC.W    STS_14-STS_TABLE        Error status register
000001AA 01A8                 DC.W    STS_15-STS_TABLE        Current Xon Character
000001AC 01AE                 DC.W    STS_16-STS_TABLE        Current Xoff Character
000001AE 01B4                 DC.W    STS_17-STS_TABLE        Current ENQ Character
000001B0 01BA                 DC.W    STS_18-STS_TABLE        Current ACK Character
000001B2 01C0                 DC.W    STS_19-STS_TABLE        Current FE/PE convert Character
```

```
000001B4 01C6              DC.W    STS_20-STS_TABLE        Ignore FE/PE

                0000 01B6 STS_ERROR       EQU *
000001B6 7013                   MOVEQ   #BAD_RDS,D0
000001B8 6000 0674              BRA     IOESCAPE                error number is passed in d0
                          *-------------------------------------------------------------------------
                0000 01BC STS_0           EQU *                   -- ID register --
000001BC 16A9 0001              MOVE.B  ID_REG(A1),(A3)         get id from card
000001C0 4E75                   RTS
                          *-------------------------------------------------------------------------
                0000 01C2 STS_1           EQU *                   -- Interrupt Status --
000001C2 16A9 0003              MOVE.B  INTR_SW(A1),(A3)        get result from card
000001C6 4E75                   RTS
                          *-------------------------------------------------------------------------
                0000 01C8 STS_2           EQU *                   -- "Busy Bits" --
000001C8 1E29 0003              MOVE.B  INTR_SW(A1),D7          --> interrupt enabled bit (1)
000001CC CE3C 0080              AND.B   #$80,D7
000001D0 EC0F                   LSR.B   #6,D7                   move it to correct position

000001D2 4AAA 0024              TST.L   BUFI_OFF(A2)            --> transfer active bit (0)
000001D6 6606                   BNE.S   SET_BIT_0
000001D8 4AAA 0028              TST.L   BUFO_OFF(A2)
000001DC 6704                   BEQ.S   DO_BIT_4                neither transfer is active

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

                0000 01E2 DO_BIT_4        EQU *                   --> not transmitting bit (4)
000001E2 4A2A 003D              TST.B   XMITTING(A2)
000001E6 6604                   BNE.S   DO_BIT_5
000001E8 08C7 0004              BSET    #4,D7

                0000 01EC DO_BIT_5        EQU *                   --> not receiving bit (5)
000001EC 4A2A 003C              TST.B   RECEIVING(A2)
000001F0 6604                   BNE.S   END_STS_2
000001F2 08C7 0005              BSET    #5,D7

                0000 01F6 END_STS_2       EQU *                   store away result
000001F6 1687                   MOVE.B  D7,(A3)
000001F8 4E75                   RTS
                          *-------------------------------------------------------------------------
                0000 01FA STS_3           EQU *                   -- Baud rate --
000001FA 538B                   SUBQ.L  #1,A3                   this routine returns a word
                          *
                          *   Get divisor (a critical section since it uses DLAB)
                          *
000001FC 1E29 0003              MOVE.B  INTR_SW(A1),D7          save card interrupt status
00000200 4229 0003              CLR.B   INTR_SW(A1)             disable interrupts

00000204 08E9 0007              BSET    #7,LINE_CONT(A1)        get access to divisor latches
                0017
0000020A 1029 0013              MOVE.B  DIV1(A1),D0             get upper half of divisor
0000020E E148                   LSL.W   #8,D0
00000210 1029 0011              MOVE.B  DIV0(A1),D0             get lower half of divisor
00000214 08A9 0007              BCLR    #7,LINE_CONT(A1)        reset DLAB so normal operation can resume
                0017

```



```
0000021A 1347 0003              MOVE.B  D7,INTR_SW(A1)          restore interrupts
                          *
                          *   Check for special divisors which division is inexact
                          *
0000021E 4A40                   TST.W   D0                      - infinite baud rate ? -
00000220 6602                     BNE.S   IS85
00000222 4E75                     RTS                           return zero baud rate
00000224 B07C 0055 IS85         CMP.W   #85,D0                  - 1800 baud ? -
00000228 6606                     BNE.S   IS77
0000022A 36BC 0708                MOVE.W  #1800,(A3)
0000022E 4E75                     RTS
00000230 B07C 004D IS77         CMP.W   #77,D0                  - 2000 baud ? -
00000234 6606                     BNE.S   IS43
00000236 36BC 07D0                MOVE.W  #2000,(A3)
0000023A 4E75                     RTS
0000023C B07C 002B IS43         CMP.W   #43,D0                  - 3600 baud ? -
00000240 6606                     BNE.S   IS21
00000242 36BC 0E10                MOVE.W  #3600,(A3)
00000246 4E75                     RTS
00000248 B07C 0015 IS21         CMP.W   #21,D0                  - 7200 baud ? -
0000024C 6606                     BNE.S   REGULAR
0000024E 36BC 1C20                MOVE.W  #7200,(A3)
00000252 4E75                     RTS
                0000 0254 REGULAR         EQU *
                          *
                          *   Compute baud rate by:  baud_rate = (freq/16) / divisor
                          *
00000254 223C 0002              MOVE.L  #153600,D1
                5800
0000025A 6100 05E8              BSR     RDIVU                   do the division
0000025E 3681                   MOVE.W  D1,(A3)                 store away the answer (d1)
00000260 4E75                   RTS
                          *-------------------------------------------------------------------------
                0000 0262 STS_4           EQU *                   -- Character control --
00000262 1E29 0017              MOVE.B  LINE_CONT(A1),D7        get line control
00000266 CE3C 003F              AND.B   #$3F,D7                 remove top two bits

0000026A 1C2A 0040              MOVE.B  S_HANDSH(A2),D6         get handshake
0000026E EB0E                   LSL.B   #5,D6                   move it to top two bits

00000270 8E06                   OR.B    D6,D7                   combine to form register result
00000272 1687                   MOVE.B  D7,(A3)
00000274 4E75                   RTS
                          *-------------------------------------------------------------------------
                0000 0276 STS_5           EQU *                   -- Modem control --
00000276 16A9 0019              MOVE.B  MODEM_CONT(A1),(A3)     read directly from the UART
0000027A 4E75                   RTS
                          *-------------------------------------------------------------------------
                0000 027C STS_6           EQU *                   -- Data in --
0000027C 6100 065A              BSR     CHECK_ERROR             check for errors trapped by ISRs

00000280 6100 07EE              BSR     QUEUE_EMPTY             read from buffer if not empty
00000284 6714                   BEQ.S   READ_UART               else read directly from UART

00000286 1A29 0003              MOVE.B  INTR_SW(A1),D5          save interrupt state
0000028A 4229 0003              CLR.B   INTR_SW(A1)             disable card interrupts for critical section

```

```
PAGE  23 [3.0] 12/26/84 21:34:08 RS232 DRIVERS -- STATUS                                  *** File name: RS ***

  918     0000028E 6100 0688          BSR     GET_CHAR                (get character with handshake)
  919     00000292 1682               MOVE.B  D2,(A3)
  920
  921     00000294 1345 0003          MOVE.B  D5,INTR_SW(A1)          restore interrupt state
  922     00000298 4E75               RTS
  923               0000 029A READ_UART       EQU *
  924     0000029A 16A9 0011          MOVE.B  DATA(A1),(A3)
  925     0000029E 4E75               RTS
  926                         *-------------------------------------------------------------------------------
  927               0000 02A0 STS_7           EQU *                   -- Optional circuits --
  928     000002A0 7E00               MOVEQ   #0,D7                   RETURN 0 IF 98644
  929     000002A2 1C29 0001          MOVE.B  ID_REG(A1),D6           GET ID REG
  930     000002A6 0886 0007          BCLR    #7,D6                   CLEAR REMOTE BIT
  931     000002AA BC3C 0042          CMP.B   #66,D6                  BRANCH IF 98644
  932     000002AE 6706               BEQ.S   STS_7B
  933     000002B0 1E29 0005          MOVE.B  BAUD_SW(A1),D7          read from the card hardware
  934     000002B4 E80F               LSR.B   #4,D7                   right justify (and get rid of baud info)
  935     000002B6 1687       STS_7B  MOVE.B  D7,(A3)
  936     000002B8 4E75               RTS
  937                         *-------------------------------------------------------------------------------
  938               0000 02BA STS_8           EQU *                   -- interrupt enable mask --
  939     000002BA 16A9 0013          MOVE.B  INTR_EN(A1),(A3)        read directly from the UART
  940     000002BE 4E75               RTS
  941                         *-------------------------------------------------------------------------------
  942               0000 02C0 STS_9           EQU *                   -- interrupt cause --
  943     000002C0 16A9 0015          MOVE.B  INTR_ID(A1),(A3)        read directly from the UART
  944     000002C4 4E75               RTS
  945                         *-------------------------------------------------------------------------------
  946               0000 02C6 STS_10          EQU *                   -- UART status --
  947     000002C6 1E29 0003          MOVE.B  INTR_SW(A1),D7          save card interrupt condition
  948     000002CA 4229 0003          CLR.B   INTR_SW(A1)             disable interrupts for critical section
  949
  950     000002CE 1C2A 003F          MOVE.B  S_LINE(A2),D6           get accumulated line status
  951     000002D2 422A 003F          CLR.B   S_LINE(A2)              reset it since it is read destructive
  952
  953     000002D6 1347 0003          MOVE.B  D7,INTR_SW(A1)          restore interrupts (end critical section)
  954
  955     000002DA CC3C 001E          AND.B   #$1E,D6                 only use the read destructive bits
  956     000002DE 8C29 001B          OR.B    LINE_STAT(A1),D6        combine it with the current status
  957
  958     000002E2 6100 078C          BSR     QUEUE_EMPTY             use internal buffer to determine bit 0
  959     000002E6 6704               BEQ.S   DONT_SET
  960     000002E8 08C6 0000          BSET    #0,D6                   set receive buffer full bit
  961               0000 02EC DONT_SET        EQU *
  962     000002EC 1686               MOVE.B  D6,(A3)
  963     000002EE 4E75               RTS
  964                         *-------------------------------------------------------------------------------
  965               0000 02F0 STS_11          EQU *                   -- modem status --
  966     000002F0 1E29 0003          MOVE.B  INTR_SW(A1),D7          save card interrupt condition
  967     000002F4 4229 0003          CLR.B   INTR_SW(A1)             disable interrupts for critical section
  968
  969     000002F8 1C2A 003E          MOVE.B  S_MODEM(A2),D6          get accumulated copy of modem status
  970     000002FC 422A 003E          CLR.B   S_MODEM(A2)             clear it since it is read destructive
  971
  972     00000300 1347 0003          MOVE.B  D7,INTR_SW(A1)          restore interrupts (end critical section)
  973
  974     00000304 CC3C 000F          AND.B   #$0F,D6                 only use the read destructive bits
```

```
PAGE  24 [3.0] 12/26/84 21:34:08 RS232 DRIVERS -- STATUS                                  *** File name: RS ***

  975     00000308 8C29 001D          OR.B    MODEM_STAT(A1),D6       combine it with the current status
  976     0000030C 1686               MOVE.B  D6,(A3)
  977     0000030E 4E75               RTS
  978                         *-------------------------------------------------------------------------------
  979               0000 0310 STS_12          EQU *                   -- connect/disconnect --
  980     00000310 16AA 003A          MOVE.B  CONNECTED(A2),(A3)      get the pseudo-register
  981     00000314 4E75               RTS
  982                         *-------------------------------------------------------------------------------
  983               0000 0316 STS_13          EQU *                   -- hardware handshake register --
  984     00000316 16AA 003B          MOVE.B  MODEM_ON(A2),(A3)
  985     0000031A 4E75               RTS
  986                         *-------------------------------------------------------------------------------
  987               0000 031C STS_14          EQU *                   -- current error status --
  988     0000031C 1E29 0003          MOVE.B  INTR_SW(A1),D7          save interrupt state
  989     00000320 4229 0003          CLR.B   INTR_SW(A1)             disable interrupts
  990
  991     00000324 376A 0036          MOVE.W  S_ERROR+2(A2),-1(A3)    get (lower word of) the error
          FFFF
  992                         *                                       this is a word register!
  993     0000032A 42AA 0034          CLR.L   S_ERROR(A2)             the read is destructive
  994
  995     0000032E 1347 0003          MOVE.B  D7,INTR_SW(A1)          restore interrupt state
  996     00000332 4E75               RTS
  997                         *-------------------------------------------------------------------------------
  998               0000 0334 STS_15          EQU *                   -- Current Xon character --
  999     00000334 16AA 0041          MOVE.B  XON_CHAR(A2),(A3)
 1000     00000338 4E75               RTS
 1001                         *-------------------------------------------------------------------------------
 1002               0000 033A STS_16          EQU *                   -- Current Xoff character --
 1003     0000033A 16AA 0042          MOVE.B  XOFF_CHAR(A2),(A3)
 1004     0000033E 4E75               RTS
 1005                         *-------------------------------------------------------------------------------
 1006               0000 0340 STS_17          EQU *                   -- Current ENQ character --
 1007     00000340 16AA 0043          MOVE.B  ENQ_CHAR(A2),(A3)
 1008     00000344 4E75               RTS
 1009                         *-------------------------------------------------------------------------------
 1010               0000 0346 STS_18          EQU *                   -- Current ACK character --
 1011     00000346 16AA 0044          MOVE.B  ACK_CHAR(A2),(A3)
 1012     0000034A 4E75               RTS
 1013                         *-------------------------------------------------------------------------------
 1014               0000 034C STS_19          EQU *                   -- Current FE/PE convert character --
 1015     0000034C 16AA 0045          MOVE.B  CONV_CHAR(A2),(A3)
 1016     00000350 4E75               RTS
 1017                         *-------------------------------------------------------------------------------
 1018               0000 0352 STS_20          EQU *                   -- Ignore FE/PE (aaa)
 1019     00000352 16AA 0046          MOVE.B  IGNORE_PE(A2),(A3)                      (aaa)
 1020     00000356 4E75               RTS                                             (aaa)
```

```
1023                        ********************************************************************************
1024                        *
1025                        *       write control
1026                        *-------------------------------------------------------------------------------
1027                        *       CONVENTION:  D0.W -- value of the control
1028                        *
1029                        ********************************************************************************
1030
1031              0000 0358 RS_RS_WTC       EQU *
1032
1033                        *
1034                        *   Pascal Interface Overhead
1035                        *
1036
1037    00000358 205F               MOVEA.L (SP)+,A0                get return address
1038    0000035A 301F               MOVE.W  (SP)+,D0                get value
1039    0000035C 321F               MOVE.W  (SP)+,D1                get register number
1040    0000035E 245F               MOVEA.L (SP)+,A2                get temp address
1041    00000360 226A 0020          MOVEA.L C_ADR(A2),A1            get card address
1042    00000364 4850               PEA     (A0)                    restore return address
1043
1044                        *
1045                        *   Check for legal registers and jump to correct register handler
1046                        *
1047
1048    00000366 4A41               TST.W   D1                      check for negative register number
1049    00000368 6B3A               BMI.S   CONT_ERROR              if so goto common ioescape routine
1050    0000036A B27C 0014          CMP.W   #REG_MAX,D1             check for too large register number
1051    0000036E 6E34               BGT.S   CONT_ERROR
1052
1053    00000370 E349               LSL.W   #1,D1                   get ready for (word) table jump
1054    00000372 323B 1006          MOVE.W  CONT_TABLE(D1),D1       get offset from status table
1055    00000376 4EFB 1002          JMP     CONT_TABLE(D1)          do the indexed jump
1056
1057              0000 037A CONT_TABLE      EQU *
1058    0000037A 0030               DC.W    CONT_0-CONT_TABLE       Reset
1059    0000037C 0038               DC.W    CONT_1-CONT_TABLE       Break
1060    0000037E 002A               DC.W    CONT_ERROR-CONT_TABLE   Register 2 Undefined
1061    00000380 0062               DC.W    CONT_3-CONT_TABLE       Baud rate
1062    00000382 009E               DC.W    CONT_4-CONT_TABLE       Character control register
1063    00000384 00D2               DC.W    CONT_5-CONT_TABLE       Modem control register
1064    00000386 00D8               DC.W    CONT_6-CONT_TABLE       Data out register
1065    00000388 00E2               DC.W    CONT_7-CONT_TABLE       Optional circuits register
1066    0000038A 002A               DC.W    CONT_ERROR-CONT_TABLE   Register 8 Undefined
1067    0000038C 002A               DC.W    CONT_ERROR-CONT_TABLE   Register 9 Undefined
1068    0000038E 002A               DC.W    CONT_ERROR-CONT_TABLE   Register 10 Undefined
1069    00000390 002A               DC.W    CONT_ERROR-CONT_TABLE   Register 11 Undefined
1070    00000392 00F8               DC.W    CONT_12-CONT_TABLE      Connect/Disconnect register
1071    00000394 010C               DC.W    CONT_13-CONT_TABLE      Hardware handshake register
1072    00000396 011C               DC.W    CONT_14-CONT_TABLE      Soft reset register
1073    00000398 0126               DC.W    CONT_15-CONT_TABLE      Redefine Xon Character
1074    0000039A 012C               DC.W    CONT_16-CONT_TABLE      Redefine Xoff Character
1075    0000039C 0132               DC.W    CONT_17-CONT_TABLE      Redefine ENQ Character
1076    0000039E 0138               DC.W    CONT_18-CONT_TABLE      Redefine ACK Character
1077    000003A0 013E               DC.W    CONT_19-CONT_TABLE      Redefine FE/PE convert Character
1078    000003A2 0144               DC.W    CONT_20-CONT_TABLE      Ignore FE/PE (aaa)
1079
```



```
1080              0000 03A4 CONT_ERROR      EQU *
1081    000003A4 7013               MOVEQ   #BAD_RDS,D0
1082    000003A6 6000 0486          BRA     IOESCAPE                error number is passed in d0
1083                        *-------------------------------------------------------------------------------
1084              0000 03AA CONT_0          EQU *                   -- reset --
1085    000003AA 4A40               TST.W   D0
1086    000003AC 6600 FC5E          BNE     ASM_INIT                initialize if any bit is set
1087    000003B0 4E75               RTS
1088                        *-------------------------------------------------------------------------------
1089              0000 03B2 CONT_1          EQU *                   -- send break --
1090    000003B2 4A40               TST.W   D0
1091    000003B4 6724               BEQ.S   EXIT_C1                 no-op if control value is zero
1092                        *
1093                        *   set and hold break for 400ms
1094                        *
1095    000003B6 08E9 0006          BSET    #6,LINE_CONT(A1)        set the break bit in UART
              0017
1096    000003BC 2F3C 0006          MOVE.L  #400000,-(SP)           USE DELAY ROUTINE   ttt  JS 8/3/83
              1A80
1097              0000 03C2 WAIT_BREAK      EQU *
1098    000003C2 4EB9 0000          JSR     DELAY_TIMER             CALL DELAY ROUTINE   ttt JS 8/3/83
              0000
1099                        *
1100                        *   release break and wait 60ms for break to clear
1101                        *
1102    000003C8 08A9 0006          BCLR    #6,LINE_CONT(A1)        clear the break bit in UART
              0017
1103    000003CE 2F3C 0000          MOVE.L  #60000,-(SP)            SETUP FOR DELAY ROUTINE  ttt JS 8/3/83
              EA60
1104              0000 03D4 WAIT_BREAK2     EQU *
1105    000003D4 4EB9 0000          JSR     DELAY_TIMER             CALL DELAY ROUTINE       ttt JS 8/3/83
              0000
1106    000003DA 4E75               EXIT_C1 RTS
1107                        *-------------------------------------------------------------------------------
1108              0000 03DC CONT_3          EQU *                   -- Baud rate --
1109                        *
1110                        *   check for overflow -- a baud rate with a resulting divisor more than
1111                        *                         sixteen bits long.
1112                        *   (underflow is not checked because it cannot be generated with a word
1113                        *    length baud rate)
1114                        *
1115    000003DC B07C 0005          CMP.W   #5,D0                   5 is the lowest baud rate possible
1116    000003E0 6C00 0008          BGE     CALC_DIV
1117
1118    000003E4 7017               MOVEQ   #IO_MISC,D0             value out of range -- io misc error
1119    000003E6 6000 0446          BRA     IOESCAPE
1120                        *
1121                        *   calculate divisor :  div = (freq/16) / baud_rate
1122                        *
1123              0000 03EA CALC_DIV        EQU *
1124    000003EA 223C 0002          MOVE.L  #153600,D1              d1 := freq/16
              5800
1125    000003F0 6100 0452          BSR     RDIVU                   d1.w := d1.l / d0.w ( freq/16 / baud )
1126                        *
1127                        *   move the divisor to hardware (critical section: uses DLAB)
1128                        *
1129    000003F4 1E29 0003          MOVE.B  INTR_SW(A1),D7          save card interrupt state
```

```
 1130    000003F8 4229 0003          CLR.B   INTR_SW(A1)             disable card interrupts
 1131
 1132    000003FC 08E9 0007          BSET    #7,LINE_CONT(A1)        set DLAB to get access to divisor latches
                  0017
 1133    00000402 1341 0011          MOVE.B  D1,DIV0(A1)             set lower half of divisor latch
 1134    00000406 E049               LSR.W   #8,D1
 1135    00000408 1341 0013          MOVE.B  D1,DIV1(A1)             set upper half of divisor latch
 1136    0000040C 08A9 0007          BCLR    #7,LINE_CONT(A1)        clear DLAB for normal use
                  0017
 1137
 1138    00000412 1347 0003          MOVE.B  D7,INTR_SW(A1)          restore card interrupt state
 1139
 1140    00000416 4E75               RTS
 1141                        *-------------------------------------------------------------------------------
 1142             0000 0418 CONT_4           EQU *                   -- Character Control --
 1143    00000418 1E00               MOVE.B  D0,D7                   save a copy of control value
 1144
 1145    0000041A C03C 003F          AND.B   #$3F,D0                 use only bottom 6 bits
 1146    0000041E 1340 0017          MOVE.B  D0,LINE_CONT(A1)        for line control
 1147
 1148    00000422 CE3C 00C0          AND.B   #$C0,D7                 handshake is top two bits
 1149    00000426 EA0F               LSR.B   #5,D7                   shift it for later use
 1150    00000428 BE2A 0040          CMP.B   S_HANDSH(A2),D7         if handshake changed
 1151    0000042C 671C               BEQ.S   EXIT_C4
 1152
 1153    0000042E 1C29 0003          MOVE.B  INTR_SW(A1),D6          save interrupt state
 1154    00000432 4229 0003          CLR.B   INTR_SW(A1)             disable interrupts
 1155
 1156    00000436 1547 0040          MOVE.B  D7,S_HANDSH(A2)         save new handshake
 1157    0000043A 157C 0001          MOVE.B  #1,RECEIVING(A2)        \
                  003C
 1158    00000440 157C 0001          MOVE.B  #1,XMITTING(A2)         /   reset the handshake flags
                  003D
 1159
 1160    00000446 1346 0003          MOVE.B  D6,INTR_SW(A1)          restore interrupt state
 1161    0000044A 4E75       EXIT_C4 RTS
 1162                        *-------------------------------------------------------------------------------
 1163             0000 044C CONT_5           EQU *                   -- modem control --
 1164    0000044C 1340 0019          MOVE.B  D0,MODEM_CONT(A1)       write directly to UART
 1165    00000450 4E75               RTS
 1166                        *-------------------------------------------------------------------------------
 1167             0000 0452 CONT_6           EQU *                   -- Data out --
 1168    00000452 6100 0484          BSR     CHECK_ERROR             check for errors trapped by ISRs
 1169
 1170    00000456 1340 0011          MOVE.B  D0,DATA(A1)             write directly to UART
 1171    0000045A 4E75               RTS
 1172                        *-------------------------------------------------------------------------------
 1173             0000 045C CONT_7           EQU *                   -- Optional Circuits --
 1174    0000045C 1C29 0001          MOVE.B  ID_REG(A1),D6           GET ID REG
 1175    00000460 0886 0007          BCLR    #7,D6                   CLEAR REMOTE BIT
 1176    00000464 BC3C 0042          CMP.B   #66,D6                  IS THIS A 98644 ?
 1177    00000468 6706               BEQ.S   CONT_7R                 YES, NO OP
 1178    0000046A E908               LSL.B   #4,D0                   left justify
 1179    0000046C 1340 0005          MOVE.B  D0,BAUD_SW(A1)          write directly to card hardware
 1180    00000470 4E75       CONT_7R RTS
 1181                        *-------------------------------------------------------------------------------
 1182             0000 0472 CONT_12          EQU *                   -- connect/disconnect --
```



```
 1183    00000472 4A00               TST.B   D0                      check for d0=0
 1184    00000474 6700 0408          BEQ     DISCONNECT              disconnect will do return
 1185    00000478 B03C 0001          CMP.B   #1,D0
 1186    0000047C 6700 03DC          BEQ     CONNECT                 connect will return
 1187             0000 0480 VAL_ERR          EQU *
 1188    00000480 7017               MOVEQ   #IO_MISC,D0             illegal value for register
 1189    00000482 6000 03AA          BRA     IOESCAPE
 1190                        *-------------------------------------------------------------------------------
 1191             0000 0486 CONT_13          EQU *                   -- hardware handshake --
 1192    00000486 4A00               TST.B   D0                      check for too small value
 1193    00000488 6DF6               BLT.S   VAL_ERR                 (located in cont_12)
 1194    0000048A B03C 0001          CMP.B   #1,D0                   check for too large value
 1195    0000048E 6EF0               BGT.S   VAL_ERR
 1196    00000490 1540 003B          MOVE.B  D0,MODEM_ON(A2)         assign modem flag
 1197    00000494 4E75               RTS
 1198                        *-------------------------------------------------------------------------------
 1199             0000 0496 CONT_14          EQU *                   -- soft reset --
 1200    00000496 4A40               TST.W   D0                      any bit will do reset
 1201    00000498 6704               BEQ.S   EXIT_14                 zero value does nothing
 1202
 1203    0000049A 6100 03FA          BSR     SOFT_RESET
 1204    0000049E 4E75       EXIT_14 RTS
 1205                        *-------------------------------------------------------------------------------
 1206             0000 04A0 CONT_15          EQU *                   -- redefine Xon character --
 1207    000004A0 1540 0041          MOVE.B  D0,XON_CHAR(A2)
 1208    000004A4 4E75               RTS
 1209                        *-------------------------------------------------------------------------------
 1210             0000 04A6 CONT_16          EQU *                   -- redefine Xoff character --
 1211    000004A6 1540 0042          MOVE.B  D0,XOFF_CHAR(A2)
 1212    000004AA 4E75               RTS
 1213                        *-------------------------------------------------------------------------------
 1214             0000 04AC CONT_17          EQU *                   -- redefine ENQ character --
 1215    000004AC 1540 0043          MOVE.B  D0,ENQ_CHAR(A2)
 1216    000004B0 4E75               RTS
 1217                        *-------------------------------------------------------------------------------
 1218             0000 04B2 CONT_18          EQU *                   -- redefine ACK character --
 1219    000004B2 1540 0044          MOVE.B  D0,ACK_CHAR(A2)
 1220    000004B6 4E75               RTS
 1221                        *-------------------------------------------------------------------------------
 1222             0000 04B8 CONT_19          EQU *                   -- redefine FE/PE convert character --
 1223    000004B8 1540 0045          MOVE.B  D0,CONV_CHAR(A2)
 1224    000004BC 4E75               RTS
 1225                        *-------------------------------------------------------------------------------
 1226             0000 04BE CONT_20          EQU *                   -- Ignore PE/FE --          {aaa}
 1227    000004BE 4A00               TST.B   D0                      check for too small value {aaa}
 1228    000004C0 6D0C               BLT.S   VAL_ERR2                                          {aaa}
 1229    000004C2 B03C 0001          CMP.B   #1,D0                   check for too large value {aaa}
 1230    000004C6 6E06               BGT.S   VAL_ERR2                                          {aaa}
 1231    000004C8 1540 0046          MOVE.B  D0,IGNORE_PE(A2)        assign modem flag         {aaa}
 1232    000004CC 4E75               RTS                                                       {aaa}
 1233             0000 04CE VAL_ERR2         EQU *                                             {aaa}
 1234    000004CE 7017               MOVEQ   #IO_MISC,D0             illegal value for register{aaa}
 1235    000004D0 6000 035C          BRA     IOESCAPE                                          {aaa}
```

```
1238                        ********************************************************************************
1239                        *
1240                        *       interrupt service routine
1241                        *
1242                        ********************************************************************************
1243
1244                0000 04D4 RS_RS_ISR       EQU *
1245
1246                        *
1247                        *       pascal interface overhead
1248                        *
1249
1250    000004D4 205F               MOVEA.L (SP)+,A0                get return address
1251    000004D6 245F               MOVEA.L (SP)+,A2                get temp address
1252    000004D8 228A 0020          MOVEA.L C_ADR(A2),A1            get card address
1253    000004DC 4850               PEA     (A0)                    restore return address
1254
1255                        *
1256                        *       verify there is an interrupt
1257                        *
1258
1259    000004DE 1029 0015          MOVE.B  INTR_ID(A1),D0          get interrupt cause
1260    000004E2 0800 0000          BTST    #0,D0                   make sure an interrupt is pending
1261    000004E6 6702               BEQ.S   INTR_EXIST
1262    000004E8 4E75               RTS                             no interrupt-- return
1263                        *
1264                        *       jump to appropriate interrupt handler
1265                        *
1266
1267                0000 04EA INTR_EXIST      EQU *
1268    000004EA 157C 0001          MOVE.B  #1,IN_ISR(A2)           mark isr processing
                0038
1269
1270    000004F0 4880               EXT.W   D0
1271    000004F2 303B 0006          MOVE.W  INTR_TABLE(D0),D0       (get appropriate address)
1272    000004F6 4EFB 0002          JMP     INTR_TABLE(D0)          (jump to the proper case)
1273
1274                0000 04FA INTR_TABLE      EQU *
1275    000004FA 001A               DC.W    MODEM_INTR-INTR_TABLE       modem change interrupt
1276    000004FC 0038               DC.W    OUTPUT_INTR-INTR_TABLE      output empty interrupt
1277    000004FE 00D0               DC.W    INPUT_INTR-INTR_TABLE       input available interrupt
1278    00000500 0008               DC.W    ERROR_INTR-INTR_TABLE       error interrupt
```



```
1280                        *
1281                        *   Check for possible unexpected interrupts, if found clear the interrupt
1282                        *      (different for each type of interrupt) and disable that type of
1283                        *      interrupt.
1284
1285                0000 0502 ERROR_INTR      EQU *
1286    00000502 1029 001B          MOVE.B  LINE_STAT(A1),D0        clear the interrupt (by reading line status)
1287    00000506 812A 003F          OR.B    D0,S_LINE(A2)           save line status for user
1288    0000050A 08A9 0002          BCLR    #2,INTR_EN(A1)          disable interrupts since it should not happen
                0013
1289    00000510 6000 01C0          BRA     END_ISR
1290
1291                0000 0514 MODEM_INTR      EQU *
1292    00000514 4A2A 003B          TST.B   MODEM_ON(A2)            modem handshake on?
1293    00000518 6706               BEQ.S   ABORT_MODEM                -- no abort
1294    0000051A 2E2A 0028          MOVE.L  BUFO_OFF(A2),D7         output transfer active?
1295    0000051E 6622               BNE.S   XFER_OUT                   -- yes do transfer
1296
1297                0000 0520 ABORT_MODEM     EQU *
1298    00000520 1029 001D          MOVE.B  MODEM_STAT(A1),D0       clear the interrupt (by reading modem status)
1299    00000524 812A 003E          OR.B    D0,S_MODEM(A2)          save modem status for user
1300    00000528 08A9 0003          BCLR    #3,INTR_EN(A1)          disable interrupt so it won't happen again
                0013
1301    0000052E 6000 01A2          BRA     END_ISR
1302
1303                0000 0532 OUTPUT_INTR     EQU *
1304    00000532 2E2A 0028          MOVE.L  BUFO_OFF(A2),D7         output interrupt transfer active ?
1305    00000536 660A               BNE.S   XFER_OUT                  -- yes do transfer
1306
1307    00000538 08A9 0001          BCLR    #1,INTR_EN(A1)            -- no disable interrupt
                0013
1308    0000053E 6000 0192          BRA     END_ISR
```

```
1310                        *
1311                        *   Do the output interrupt transfer
1312                        *       (but only if the THRE and the modem lines are high)
1313                        *
1314
1315              0000 0542 XFER_OUT        EQU *
1316    00000542 2647               MOVEA.L D7,A3               a3 := buffer control block
1317    00000544 1E29 001B          MOVE.B  LINE_STAT(A1),D7    check for THRE
1318    00000548 8F2A 003F          OR.B    D7,S_LINE(A2)       save line status for user
1319    0000054C 0807 0005          BTST    #5,D7
1320    00000550 6700 0180          BEQ     END_ISR
1321
1322    00000554 4A2A 003B          TST.B   MODEM_ON(A2)
1323    00000558 6708               BEQ.S   MOVE_OUT
1324    0000055A 6100 04F2          BSR     CHECK_DSR_CTS
1325    0000055E 6600 0172          BNE     END_ISR
1326
1327              0000 0562 MOVE_OUT        EQU *
1328    00000562 4247               CLR.W   D7                  clear d7.w
1329    00000564 206B 001C          MOVEA.L TEMP_OFF(A3),A0     a0 := buffer empty pointer
1330    00000568 1E18               MOVE.B  (A0)+,D7            d7 := character
1331    0000056A 1347 0011          MOVE.B  D7,DATA(A1)         write the character
1332    0000056E 2748 001C          MOVE.L  A0,TEMP_OFF(A3)     update empty pointer
1333
1334    00000572 53AB 0010          SUBQ.L  #1,TCNT_OFF(A3)     decrement the count
1335    00000576 6F08               BLE.S   END_XOUT            count=0, transfer ends
1336    00000578 BE6B 000E          CMP.W   TCHR_OFF(A3),D7     character = term. char ?
1337    0000057C 6600 0154          BNE     END_ISR
1338
1339              0000 0580 END_XOUT        EQU *
1340    00000580 4A2B 000B          TST.B   TEND_OFF(A3)        end condition enabled ?
1341    00000584 6730               BEQ.S   CLR_XOUT
1342    00000586 4A2A 003B          TST.B   MODEM_ON(A2)        modem handshake on ?
1343    0000058A 672A               BEQ.S   CLR_XOUT
1344    0000058C 0229 00F5          ANDI.B  #$F5,INTR_EN(A1)    disable output and modem interrupts
                0013
1345              0000 0592 LOOP_LAST       EQU *
1346    00000592 1E29 001B          MOVE.B  LINE_STAT(A1),D7    wait for everything transferred
1347    00000596 8F2A 003F          OR.B    D7,S_LINE(A2)       before dropping RTS
1348    0000059A 4607               NOT.B   D7
1349    0000059C CE3C 0060          AND.B   #$60,D7
1350    000005A0 66F0               BNE.S   LOOP_LAST
1351    000005A2 08A9 0001          BCLR    #1,MODEM_CONT(A1)   drop RTS is the EOI condition
                0019
1352
1353    000005A8 6100 023A          BSR     CLEAR_XFER          clear the transfer
1354    000005AC 4EB9 0000          JSR     LOGEOT              call the eot procedure
                0000
1355    000005B2 6000 011E          BRA     END_ISR
1356
1357              0000 05B6 CLR_XOUT        EQU *
1358    000005B6 0229 00F5          ANDI.B  #$F5,INTR_EN(A1)    disable output and modem interrupts
                0013
1359    000005BC 6100 0226          BSR     CLEAR_XFER          clear the transfer
1360    000005C0 4EB9 0000          JSR     LOGEOT              call the eot procedure
                0000
1361    000005C6 6000 010A          BRA     END_ISR
```



```
1363                        *
1364                        *   Input interrupts are normally active in order to fill the internal
1365                        *       buffer
1366                        *
1367              0000 05CA INPUT_INTR      EQU *
1368    000005CA 1229 001B          MOVE.B  LINE_STAT(A1),D1
1369    000005CE 1429 0011          MOVE.B  DATA(A1),D2         get the input byte (clears interrupt)
1370    000005D2 832A 003F          OR.B    D1,S_LINE(A2)       preserve line status for user
1371
1372    000005D6 4A2A 003B          TST.B   MODEM_ON(A2)
1373    000005DA 6712               BEQ.S   CHECK_BREAK         skip modem stuff if handshake off
1374                        *
1375                        *   check for both CD and DSR
1376                        *
1377    000005DC 1029 001D          MOVE.B  MODEM_STAT(A1),D0
1378    000005E0 812A 003E          OR.B    D0,S_MODEM(A2)      preserve modem status for user
1379    000005E4 4640               NOT     D0
1380    000005E6 0240 00A0          ANDI    #$A0,D0             mask appropriate bits
1381    000005EA 6600 00CA          BNE     INPUT_END           if zero then they were set previously
1382                        *
1383                        *   ignore character if break received
1384                        *
1385              0000 05EE CHECK_BREAK     EQU *
1386    000005EE 0801 0004          BTST    #4,D1               if break received,
1387    000005F2 6600 00C2          BNE     INPUT_END           then ignore character
1388                        *
1389                        *   convert Framing and Parity errors to specified character
1390                        *
1391    000005F6 1001               MOVE.B  D1,D0               save line status for later use
1392    000005F8 0240 000C          ANDI    #$0C,D0             check for PARITY and FRAMING errors
1393    000005FC 670A               BEQ.S   NO_CONVERT
1394    000005FE 4A2A 0046          TST.B   IGNORE_PE(A2)       {aaa}
1395    00000602 6604               BNE.S   NO_CONVERT          {aaa}
1396    00000604 142A 0045          MOVE.B  CONV_CHAR(A2),D2    convert the character
1397
1398              0000 0608 NO_CONVERT      EQU *
1399                        *
1400                        *       jump to appropriate handshake handler
1401                        *
1402    00000608 102A 0040          MOVE.B  S_HANDSH(A2),D0
1403    0000060C 4880               EXT.W   D0
1404    0000060E 303B 0006          MOVE.W  HAND_TABLE(PC),D0   get address to jump
1405    00000612 4EFB 0002          JMP     HAND_TABLE(PC)
1406
1407              0000 0616 HAND_TABLE      EQU *
1408    00000616 0068               DC.W    ENQ_HAND-HAND_TABLE
1409    00000618 0008               DC.W    XON_HAND-HAND_TABLE
1410    0000061A 0096               DC.W    NO_HAND-HAND_TABLE
1411    0000061C 0096               DC.W    NO_HAND-HAND_TABLE
```

```
1413                0000 061E XON_HAND        EQU *
1414                          *
1415                          *   Do host part of the handshake -- check for Xon and Xoff
1416                          *
1417    0000061E B42A 0041          CMP.B   XON_CHAR(A2),D2
1418    00000622 6626               BNE.S   CHECK_XOFF
1419    00000624 157C 0001          MOVE.B  #1,XMITTING(A2)         turn transmitting back on
                 003D
1420
1421    0000062A 4AAA 0028          TST.L   BUFO_OFF(A2)            \
1422    0000062E 6700 0086          BEQ     INPUT_END               \
1423    00000632 08E9 0001          BSET    #1,INTR_EN(A1)           \  enable interrupts if output
                 0013
1424    00000638 4A2A 003B          TST.B   MODEM_ON(A2)             /  transfer is active
1425    0000063C 6700 0078          BEQ     INPUT_END               /
1426    00000640 08E9 0003          BSET    #3,INTR_EN(A1)          /
                 0013
1427    00000646 6000 006E          BRA     INPUT_END
1428
1429                0000 064A CHECK_XOFF      EQU *
1430    0000064A B42A 0042          CMP.B   XOFF_CHAR(A2),D2
1431    0000064E 660E               BNE.S   TERM_HAND
1432    00000650 422A 003D          CLR.B   XMITTING(A2)            turn transmitting off
1433
1434    00000654 137C 0001          MOVE.B  #$1,INTR_EN(A1)         turn any possible output interrupts off
                 0013
1435    0000065A 6000 005A          BRA     INPUT_END
1436
1437                          *
1438                          *   Do terminal part of the handshake
1439                          *
1440                0000 065E TERM_HAND       EQU *
1441    0000065E 4A2A 003C          TST.B   RECEIVING(A2)           if receiving is on, might have
1442    00000662 6748               BEQ.S   PUTINQ                    to turn it off
1443    00000664 6100 0468          BSR     QUEUE_SPACE             d3 := space left in queue
1444    00000668 B67C 0028          CMP.W   #XOFF_SIZE,D3
1445    0000066C 6C3E               BGE.S   PUTINQ
1446                          *
1447                          *   Have to turn receiving off
1448                          *
1449    0000066E 162A 0042          MOVE.B  XOFF_CHAR(A2),D3        prepare to send Xoff
1450    00000672 6100 0320          BSR     SEND                    send character
1451    00000676 6634               BNE.S   PUTINQ                  send did not succeed
1452    00000678 422A 003C          CLR.B   RECEIVING(A2)           no longer expecting input
1453    0000067C 602E               BRA.S   PUTINQ                  but put present char in queue
1454
```



```
1456
1457                0000 067E ENQ_HAND        EQU *
1458    0000067E B42A 0043          CMP.B   ENQ_CHAR(A2),D2          IF char <> ENQ
1459    00000682 6600 0028          BNE     PUTINQ                     THEN put char in queue
1460    00000686 6100 0446          BSR     QUEUE_SPACE                ELSE
1461    0000068A B67C 0050          CMP.W   #ACK_SIZE,D3                 IF queue_space <= 80
1462    0000068E 6C08               BGE.S   SEND_ACK
1463    00000690 422A 003C          CLR.B   RECEIVING(A2)                    receiving := false
1464    00000694 6000 0020          BRA     INPUT_END
1465
1466                0000 0698 SEND_ACK        EQU *
1467    00000698 162A 0044          MOVE.B  ACK_CHAR(A2),D3         set up parameter in D3
1468    0000069C 6100 02FE          BSR     SEND                    send ACK
1469    000006A0 6700 0014          BEQ     INPUT_END
1470
1471    000006A4 422A 003C          CLR.B   RECEIVING(A2)           not receiving since ACK not sent
1472    000006A8 6000 000C          BRA     INPUT_END
1473
1474                          *
1475                          *   Put character (d2) in queue, and check for overrun.
1476                          *
1477                0000 06AC NO_HAND         EQU *
1478                0000 06AC PUTINQ          EQU *
1479    000006AC 6100 03CC          BSR     QUEUE_FULL      IF queue_full
1480    000006B0 670A               BEQ.S   OVERRUN            THEN overrun_error
1481    000006B2 6100 03E2          BSR     INQUEUE            ELSE inqueue(char)
1482
1483                0000 06B6 INPUT_END       EQU *
1484    000006B6 0241 0002          ANDI    #$02,D1         check for overrun error
1485    000006BA 6708               BEQ.S   CHECK_XIN
1486
1487                0000 06BC OVERRUN         EQU *
1488    000006BC 257C 0000          MOVE.L  #OVERRUN_ERROR,S_ERROR(A2)
                 013A 0034
1489
1490                          *
1491                          *   If transfer in is active then do the transfer
1492                          *
1493                0000 06C4 CHECK_XIN       EQU *
1494    000006C4 4A2A 0039          TST.B   XIN_ACT(A2)             check for transfer active
1495    000006C8 6708               BEQ.S   END_ISR
1496
1497    000006CA 266A 0024          MOVEA.L BUFI_OFF(A2),A3         a3 := buffer control block pointer
1498    000006CE 6100 00D8          BSR     DUMP_BUFFER
1499
1500                0000 06D2 END_ISR         EQU *
1501    000006D2 422A 0038          CLR.B   IN_ISR(A2)              not in isr any longer
1502    000006D6 4E75               RTS
```

```
1505                        ********************************************************************************
1506                        *
1507                        *       transfer
1508                        *
1509                        ********************************************************************************
1510
1511             0000 06D8 RS_RS_TFR       EQU *
1512
1513                        *
1514                        *   Pascal interface overhead
1515                        *
1516    000006D8 205F               MOVEA.L (SP)+,A0                get return address
1517    000006DA 265F               MOVEA.L (SP)+,A3                get buffer control block address
1518    000006DC 245F               MOVEA.L (SP)+,A2                get temp address
1519    000006DE 226A 0020          MOVEA.L C_ADR(A2),A1            get card address
1520    000006E2 4850               PEA     (A0)                    restore return address
1521                        *
1522                        *   Card overhead
1523                        *
1524    000006E4 6100 0174          BSR     CONNECT
1525    000006E8 6100 01EE          BSR     CHECK_ERROR
1526                        *
1527                        *   Check for unsupported transfer modes
1528                        *       ( done by table jump also )
1529                        *
1530    000006EC 4A2B 000A          TST.B   T_BW_OFF(A3)            word mode?
1531    000006F0 662E               BNE.S   WORD_ERR                  -- is unsupported
1532
1533    000006F2 122B 0009          MOVE.B  TUSR_OFF(A3),D1         d1.w := offset into transfer table
1534    000006F6 4881               EXT.W   D1
1535    000006F8 D241               ADD.W   D1,D1                   d1.w := word offset into table
1536    000006FA 323B 1006          MOVE.W  XFER_TABLE(D1),D1
1537    000006FE 4EFB 1002          JMP     XFER_TABLE(D1)
1538
1539             0000 0702 XFER_TABLE      EQU *
1540    00000702 0014               DC.W    XFER_ERR-XFER_TABLE     not used
1541    00000704 0014               DC.W    DMA_ERR-XFER_TABLE      serial DMA -- not supported
1542    00000706 0028               DC.W    SER_FHS-XFER_TABLE      serial FHS
1543    00000708 0028               DC.W    SER_FHS-XFER_TABLE      serial fastest -- same as serial FHS
1544    0000070A 0014               DC.W    XFER_ERR-XFER_TABLE     not used
1545
1546    0000070C 0036               DC.W    INTR_XFER-XFER_TABLE    overlap INTR
1547    0000070E 0014               DC.W    DMA_ERR-XFER_TABLE      overlap DMA -- not supported
1548    00000710 0014               DC.W    XFER_ERR-XFER_TABLE     overlap FHS -- not supported
1549    00000712 0036               DC.W    INTR_XFER-XFER_TABLE    overlap FASTEST -- same as overlap INTR
1550    00000714 0036               DC.W    INTR_XFER-XFER_TABLE    overlap overlap -- same as overlap INTR
```



```
1552                        *
1553                        *   Error escapes
1554                        *
1555             0000 0716 END_ERR         EQU *
1556             0000 0716 DMA_ERR         EQU *
1557             0000 0716 XFER_ERR        EQU *
1558    00000716 6100 00CC          BSR     CLEAR_XFER
1559    0000071A 7007               MOVEQ   #TFR_ERR,D0
1560    0000071C 6000 0110          BRA     IOESCAPE
1561
1562             0000 0720 WORD_ERR        EQU *
1563
1564    00000720 6100 00C2          BSR     CLEAR_XFER
1565    00000724 700E               MOVEQ   #NO_WORD,D0
1566    00000726 6000 0106          BRA     IOESCAPE
1567                        *
1568                        *   Set the actual mode for transfers
1569                        *
1570             0000 072A SER_FHS         EQU *
1571    0000072A 177C 0004          MOVE.B  #TT_FHS,TACT_OFF(A3)    set the actual mode
                 0007
1572    00000730 4A2B 000D          TST.B   TDIR_OFF(A3)            jump to correct direction handler
1573    00000734 6634               BNE.S   OUTPUT_XFER
1574    00000736 600C               BRA.S   INPUT_XFER
1575
1576             0000 0738 INTR_XFER       EQU *
1577    00000738 177C 0001          MOVE.B  #TT_INT,TACT_OFF(A3)    set actual mode to INTR
                 0007
1578    0000073E 4A2B 000D          TST.B   TDIR_OFF(A3)            jump to correct direction handler
1579    00000742 6626               BNE.S   OUTPUT_XFER
1580                        *       BRA.S   INPUT_XFER
1581
1582                        *
1583                        *   Input transfer setup.
1584                        *
1585
1586             0000 0744 INPUT_XFER      EQU *
1587    00000744 4A2B 000B          TST.B   TEND_OFF(A3)            end condition not allowed on input xfers
1588    00000748 66CC               BNE.S   END_ERR
1589
1590    0000074A 6100 005C          BSR     DUMP_BUFFER             do most of transfers with intr enabled
1591    0000074E 6756               BEQ.S   EXIT_TFR                if transfer done, then exit
1592
1593    00000750 4229 0003          CLR.B   INTR_SW(A1)             disable interrupts for critical section
1594    00000754 157C 0001          MOVE.B  #1,XIN_ACT(A2)
                 0039
1595    0000075A 6100 00A4          BSR     SET_XFER                set interface busy
1596    0000075E 6100 0048          BSR     DUMP_BUFFER             make sure that buffer is empty (prevent deadlock)
1597                        *                                       if eot the following code will exit
1598                        *                                       so no explicit exit is done
1599    00000762 08E9 0007          BSET    #7,INTR_SW(A1)          end of critical section
                 0003
1600
1601    00000768 6026               BRA.S   CHECK_FHS               end of input transfer setup
```

```
1603                        *
1604                        *   Output transfer setup
1605                        *
1606
1607                0000 076A OUTPUT_XFER     EQU *
1608
1609    0000076A 4229 0003          CLR.B   INTR_SW(A1)             disable interrupts for critical section
1610
1611    0000076E 6100 0090          BSR     SET_XFER                set interface busy
1612    00000772 08E9 0001          BSET    #1,MODEM_CONT(A1)       set RTS
                 0019
1613    00000778 4A2A 003D          TST.B   XMITTING(A2)            if not transmitting, don't enable interrupts
1614    0000077C 6712               BEQ.S   CHECK_FHS
1615    0000077E 08E9 0001          BSET    #1,INTR_EN(A1)          enable output interrupts
                 0013
1616    00000784 4A2A 003B          TST.B   MODEM_ON(A2)            if modem handshake
1617    00000788 6706               BEQ.S   CHECK_FHS
1618    0000078A 08E9 0003          BSET    #3,INTR_EN(A1)          enable modem interrupts
                 0013
1619
1620                        *
1621                        *   IF serial transfer THEN wait until transfer is done
1622                        *
1623                0000 0790 CHECK_FHS       EQU *
1624    00000790 08E9 0007          BSET    #7,INTR_SW(A1)          end of critical section
                 0003
1625
1626    00000796 0C2B 0004          CMPI.B  #TT_FHS,TACT_OFF(A3)
                 0007
1627    0000079C 6608               BNE.S   EXIT_TFR
1628                0000 079E WAIT_FHS        EQU *
1629    0000079E 0C2B 00FF          CMPI.B  #255,T_SC_OFF(A3)       wait until buffer is not busy
                 0005
1630    000007A4 66F8               BNE.S   WAIT_FHS
1631                0000 07A6 EXIT_TFR        EQU *
1632    000007A6 4E75               RTS
```



```
1635                        ********************************************************************************
1636                        *
1637                        *       TRANSFER SUPPORT ROUTINES
1638                        *
1639                        ********************************************************************************
1640                        *
1641                        *       DUMP BUFFER
1642                        *          transfer from the internal queue to user queue
1643                        *
1644                        *       ON ENTRY: a3 - points to buffer control block
1645                        *       ON EXIT : IF transfer was completed
1646                        *                    THEN d2.L=0 and Z=1
1647                        *                    ELSE d2.L=1 and Z=0
1648                        *       USES:     a0 - current fill pointer to user input buffer
1649                        *                 d2 - character being transfered
1650                        *-------------------------------------------------------------------------------
1651
1652                0000 07A8 DUMP_BUFFER     EQU *
1653    000007A8 206B 0020          MOVEA.L TFIL_OFF(A3),A0         get fill pointer
1654    000007AC 4242               CLR.W   D2                      clear top half of D2 (for later compares)
1655
1656                0000 07AE DUMP_LOOP       EQU *
1657    000007AE 6100 02C0          BSR     QUEUE_EMPTY
1658    000007B2 6728               BEQ.S   EXIT_DUMP
1659    000007B4 6100 0162          BSR     GET_CHAR                d2 := character
1660    000007B8 10C2               MOVE.B  D2,(A0)+                put it in the linear buffer
1661    000007BA 53AB 0010          SUBQ.L  #1,TCNT_OFF(A3)         decrement count
1662    000007BE 6F06               BLE.S   END_XIN
1663    000007C0 B46B 000E          CMP.W   TCHR_OFF(A3),D2
1664    000007C4 66E8               BNE.S   DUMP_LOOP
1665
1666                0000 07C6 END_XIN         EQU *
1667    000007C6 2748 0020          MOVE.L  A0,TFIL_OFF(A3)         update fill pointer
1668    000007CA 422A 0039          CLR.B   XIN_ACT(A2)
1669    000007CE 6100 0014          BSR     CLEAR_XFER
1670    000007D2 4EB9 0000          JSR     LOGEOT
                 0000
1671
1672    000007D8 4282               CLR.L   D2                      set Z flag to mark transfer ended
1673    000007DA 4E75               RTS
1674
1675                0000 07DC EXIT_DUMP       EQU *
1676    000007DC 2748 0020          MOVE.L  A0,TFIL_OFF(A3)         update fill pointer
1677
1678    000007E0 7401               MOVEQ   #1,D2                   clear Z flage to mark transfer still active
1679    000007E2 4E75               RTS
```

```
PAGE  39 [3.0] 12/26/84 21:34:08 RS232 DRIVERS -- TRANSFER SUPPORT ROUTINES                *** File name: RS ***

1681                        *-------------------------------------------------------------------------------
1682                        *       CLEAR_XFER
1683                        *          make a transfer inactive (unlink temp space and buffer control block)
1684                        *       ON ENTRY:  a3 - points to the buffer control block
1685                        *-------------------------------------------------------------------------------
1686
1687              0000 07E4 CLEAR_XFER      EQU *
1688    000007E4 422B 0007          CLR.B   TACT_OFF(A3)              clear actual transfer mode
1689    000007E8 177C 00FF          MOVE.B  #255,T_SC_OFF(A3)         set the buffer not busy
                 0005
1690    000007EE 4A2B 000D          TST.B   TDIR_OFF(A3)
1691    000007F2 6606               BNE.S   CLEAR_OUT
1692    000007F4 42AA 0024          CLR.L   BUFI_OFF(A2)              clear input transfer
1693    000007F8 4E75               RTS
1694              0000 07FA CLEAR_OUT       EQU *
1695    000007FA 42AA 0028          CLR.L   BUFO_OFF(A2)              clear output transfer
1696    000007FE 4E75               RTS
1697
1698                        *-------------------------------------------------------------------------------
1699                        *       SET_XFER
1700                        *          make a transfer active (link temp space with buffer control block)
1701                        *       ON ENTRY:  a3 - the buffer control block
1702                        *-------------------------------------------------------------------------------
1703
1704              0000 0800 SET_XFER        EQU *
1705    00000800 176A 002D          MOVE.B  IO_SC(A2),T_SC_OFF(A3)    set the buffer busy
                 0005
1706    00000806 4A2B 000D          TST.B   TDIR_OFF(A3)
1707    0000080A 6606               BNE.S   SET_OUT
1708    0000080C 254B 0024          MOVE.L  A3,BUFI_OFF(A2)           set sc's input active
1709    00000810 4E75               RTS
1710              0000 0812 SET_OUT         EQU *
1711    00000812 254B 0028          MOVE.L  A3,BUFO_OFF(A2)           set sc's output active
1712    00000816 4E75               RTS
1713
1714                        *-------------------------------------------------------------------------------
1715                        *       CHECK_XFER_IN, CHECK_XFER_OUT
1716                        *          gives an error if a transfer is active
1717                        *       USES:  d0 -- only if an ioescape is to be given
1718                        *-------------------------------------------------------------------------------
1719
1720              0000 0818 CHECK_XFER_IN   EQU *
1721    00000818 4AAA 0024          TST.L   BUFI_OFF(A2)
1722    0000081C 660A               BNE.S   BUSY_ERR
1723    0000081E 4E75               RTS
1724
1725              0000 0820 CHECK_XFER_OUT  EQU *
1726    00000820 4AAA 0028          TST.L   BUFO_OFF(A2)
1727    00000824 6602               BNE.S   BUSY_ERR
1728    00000826 4E75               RTS
1729
1730              0000 0828 BUSY_ERR        EQU *
1731    00000828 7008               MOVEQ   #SC_BUSY,D0
1732    0000082A 6000 0002          BRA     IOESCAPE
```

```
PAGE  40 [3.0] 12/26/84 21:34:08 RS232 DRIVERS -- COMMON UTILITIES                        *** File name: RS ***

1735                        ********************************************************************************
1736                        *
1737                        *       Useful Subroutines
1738                        *
1739                        ********************************************************************************
1740                        *
1741                        *       IOESCAPE
1742                        *          ON ENTRY:  d0.L -- contains the escape code
1743                        *-------------------------------------------------------------------------------
1744
1745              0000 082E IOESCAPE        EQU *
1746    0000082E 2B40 FFBE          MOVE.L  D0,IOE_RSLT(A5)           *  put ioe_result
1747    00000832 4280               CLR.L   D0                        *<<< BUG FIX >>>
1748    00000834 102A 002D          MOVE.B  IO_SC(A2),D0              *  get select code of card
1749    00000838 2B40 FFBA          MOVE.L  D0,IOE_SC(A5)             *  put ioe_sc
1750    0000083C 3B7C FFE6          MOVE.W  #IOE_ERROR,ESC_CODE(A5)   *  escapecode := ioe_error
                 FFFE
1751    00000842 4E4A               TRAP    #10                       *  do Pascal escape
1752
1753                        *-------------------------------------------------------------------------------
1754                        *       RDIVU
1755                        *          unsigned integer divide rounded.
1756                        *          ON ENTRY:  d0.w -- divisor (unchanged by this routine)
1757                        *                     d1.l -- dividend
1758                        *          ON EXIT:   d1.w -- rounded quotient
1759                        *-------------------------------------------------------------------------------
1760              0000 0844 RDIVU           EQU *
1761    00000844 82C0               DIVU    D0,D1                     do truncated division
1762    00000846 4841               SWAP    D1                        get access to remainder
1763    00000848 E349               LSL.W   #1,D1                     multiply remainder by 2
1764    0000084A 6508               BCS.S   ROUND                     if carry then remainder*2>divisor
1765    0000084C B041               CMP.W   D1,D0                     remainder*2 > divisor ?
1766    0000084E 6F04               BLE.S   ROUND                     round up if so.
1767                        *                                         --do not round --
1768    00000850 4841               SWAP    D1                        get quotient
1769    00000852 4E75               RTS
1770              0000 0854 ROUND           EQU *                     --round up--
1771    00000854 4841               SWAP    D1                        get old quotient
1772    00000856 5241               ADDQ.W  #1,D1                     increment (do the rounding)
1773    00000858 4E75               RTS
```

```
1775                        *-------------------------------------------------------------------------------
1776                        *       CONNECT
1777                        *          connects the card if not connected already.
1778                        *
1779                        *       uses : d6,d7 by called routines
1780                        *-------------------------------------------------------------------------------
1781
1782             0000 085A CONNECT         EQU *
1783    0000085A 4A2A 003A                 TST.B   CONNECTED(A2)           IF connected THEN do nothing
1784    0000085E 661C                      BNE.S   EXIT_CONNECT
1785
1786    00000860 08E9 0000                 BSET    #0,MODEM_CONT(A1)       set DTR
                 0019
1787    00000866 6100 002E                 BSR     SOFT_RESET              initialize the dynamic data
1788    0000086A 137C 0001                 MOVE.B  #1,INTR_EN(A1)          enable receive interrupts
                 0013
1789    00000870 157C 0001                 MOVE.B  #1,CONNECTED(A2)        set connected
                 003A
1790    00000876 08E9 0007                 BSET    #7,INTR_SW(A1)          enable card interrupts
                 0003
1791             0000 087C EXIT_CONNECT    EQU *
1792    0000087C 4E75                      RTS
1793
1794
1795                        *-------------------------------------------------------------------------------
1796                        *       DISCONNECT
1797                        *          disconnect and disable interrupts
1798                        *-------------------------------------------------------------------------------
1799
1800             0000 087E DISCONNECT      EQU *
1801    0000087E 08A9 0007                 BCLR    #7,INTR_SW(A1)          disable card interrupts
                 0003
1802    00000884 422A 003A                 CLR.B   CONNECTED(A2)           set disconnected
1803    00000888 0229 00FC                 ANDI.B  #$FC,MODEM_CONT(A1)     drop DTR and RTS
                 0019
1804    0000088E 4E71                      NOP
1805    00000890 4229 0013                 CLR.B   INTR_EN(A1)             disable all UART interrupts
1806    00000894 4E75                      RTS
```



```
1808                        *-------------------------------------------------------------------------------
1809                        *       SOFT_RESET
1810                        *          initialize the "dynamic" attributes of the drivers
1811                        *
1812                        *       uses :  d6,d7 as temporary
1813                        *-------------------------------------------------------------------------------
1814
1815             0000 0896 SOFT_RESET      EQU *
1816
1817    00000896 4EB9 0000                 JSR     ABORT_IO                abort transfers
                 0000
1818
1819    0000089C 1C29 0003                 MOVE.B  INTR_SW(A1),D6          save interrupt state
1820    000008A0 4229 0003                 CLR.B   INTR_SW(A1)             disable interrupts
1821
1822    000008A4 0229 0001                 ANDI.B  #1,INTR_EN(A1)          disable modem and transmit interrupts
                 0013
1823    000008AA 6100 01B4                 BSR     INIT_QUEUE
1824    000008AE 1E29 0011                 MOVE.B  DATA(A1),D7             destroy any data
1825    000008B2 422A 003F                 CLR.B   S_LINE(A2)
1826    000008B6 1E29 001B                 MOVE.B  LINE_STAT(A1),D7        reset the line status (destructive read)
1827    000008BA 422A 003E                 CLR.B   S_MODEM(A2)
1828    000008BE 1E29 001D                 MOVE.B  MODEM_STAT(A1),D7       reset the modem status (destructive read)
1829
1830
1831    000008C2 42AA 0034                 CLR.L   S_ERROR(A2)
1832    000008C6 157C 0001                 MOVE.B  #1,RECEIVING(A2)
                 003C
1833    000008CC 157C 0001                 MOVE.B  #1,XMITTING(A2)
                 003D
1834
1835    000008D2 1346 0003                 MOVE.B  D6,INTR_SW(A1)          restore the interrupt state
1836    000008D6 4E75                      RTS
1837
1838
1839                        *-------------------------------------------------------------------------------
1840                        *       CHECK_ERROR
1841                        *          check for errors recorded in interrupt service routines (ISRs)
1842                        *          USES:  D0,D7 only if doing ioescape
1843                        *-------------------------------------------------------------------------------
1844
1845             0000 08D8 CHECK_ERROR     EQU *
1846    000008D8 4AAA 0034                 TST.L   S_ERROR(A2)             is error present
1847    000008DC 6602                      BNE.S   ERROR_EXIST
1848    000008DE 4E75                      RTS                             return if not error
1849             0000 08E0 ERROR_EXIST     EQU *
1850    000008E0 1E29 0003                 MOVE.B  INTR_SW(A1),D7          save interrupt condition
1851    000008E4 4229 0003                 CLR.B   INTR_SW(A1)             disable interrupt for critical section
1852
1853    000008E8 202A 0034                 MOVE.L  S_ERROR(A2),D0          get error
1854    000008EC 42AA 0034                 CLR.L   S_ERROR(A2)             clear errors
1855
1856    000008F0 1347 0003                 MOVE.B  D7,INTR_SW(A1)          restore interrupts
1857    000008F4 6000 FF38                 BRA     IOESCAPE                do pascal escape
```

```
                        *-------------------------------------------------------------------------------
                        *       WAIT_SEND
                        *          This routine waits for the transmitting flag then sends
                        *          a character.  It escapes if SEND returns with an error.
                        *       NOTE:  this routine cannot be called by ISRs!!!
                        *       ON ENTRY:  d3.B -- character to be sent
                        *       USES    :  a4 -- used by called routines
                        *                  d4,d6,d7 -- by called routines
                        *-------------------------------------------------------------------------------

            0000 08F8 WAIT_SEND         EQU *
                        *
                        *   Wait for xmitting flag (no timeouts !!)
                        *       (the wait is important for Xon/Xoff as host)
                        *
000008F8 4A2A 003D          TST.B   XMITTING(A2)
000008FC 67FA               BEQ.S   WAIT_SEND
                        *
                        *   Send the character
                        *
            0000 08FE OK_XMIT           EQU *
000008FE 6100 0094          BSR     SEND                    send character with timeout
00000902 61D4               BSR     CHECK_ERROR             check for errors found by send
00000904 4E75               RTS


                        *-------------------------------------------------------------------------------
                        *       WAIT_GET
                        *          wait until the queue is empty before getting a character
                        *       ON EXIT:  D2.B contains the character
                        *                 (the rest of D2 is not altered!)
                        *       USES:     A4.L -- parameter to WAIT
                        *                 D0,D3,D4,D6,D7 -- used by called routines
                        *-------------------------------------------------------------------------------

            0000 0906 WAIT_GET          EQU *
                        *
                        *   Wait (with timeout) for queue not empty
                        *
00000906 49FA 0138          LEA     CHECK_QUEUE,A4          \  call wait with the not queue empty
0000090A 6100 00CC          BSR     WAIT                    /  function
0000090E 61C8               BSR     CHECK_ERROR             check for wait error

00000910 6100 0006          BSR     GET_CHAR
00000914 61C2               BSR     CHECK_ERROR
00000916 4E75               RTS
```



```
                        *-------------------------------------------------------------------------------
                        *       GET_CHAR
                        *          get a character with software handshake.
                        *       ON ENTRY :  the queue is not empty!
                        *       ON EXIT:  D2.B contains the character
                        *                 (the rest of D2 is not altered!)
                        *       USES:     D3 -- space left in queue/temporary for character
                        *                 D0.W -- hardshake type & temporary
                        *                 A4,D4,D6,D7 -- temporary
                        *-------------------------------------------------------------------------------

            0000 0918 GET_CHAR          EQU *

                        *
                        *   Read the character ( and pass it back )
                        *
00000918 6100 0198          BSR     OUTQUEUE                get the character (into D2)
                        *
                        *   Check for and do handshake overhead
                        *
0000091C 4A2A 003C          TST.B   RECEIVING(A2)           if receiving
00000920 6670               BNE.S   READ_END                then no overhead needed
                        *
                        *   Jump to appropriate handshake handler
                        *
00000922 102A 0040          MOVE.B  S_HANDSH(A2),D0         get handshake
00000926 4880               EXT.W   D0
00000928 303B 0006          MOVE.W  H_TABLE(D0),D0
0000092C 4EFB 0002          JMP     H_TABLE(D0)

            0000 0930 H_TABLE           EQU *
00000930 0008               DC.W    ENQ_H-H_TABLE
00000932 0018               DC.W    XON_H-H_TABLE
00000934 0062               DC.W    READ_END-H_TABLE        no handshake (no overhead)
00000936 0062               DC.W    READ_END-H_TABLE        no handshake
```

```
1945                        *
1946                        *   ENQ/ACK handshake--send ACK if queue can handle more than 80 chars
1947                        *
1948              0000 0938 ENQ_H           EQU *
1949     00000938 6100 0194         BSR     QUEUE_SPACE             returns space left in D3
1950     0000093C B67C 0050         CMP.W   #ACK_SIZE,D3
1951     00000940 6D50              BLT.S   READ_END                space not big enough to send ACK
1952                        *
1953                        *   Send character to indicate card is receiving and set receiving flag
1954                        *
1955     00000942 162A 0044         MOVE.B  ACK_CHAR(A2),D3         send ack
1956     00000946 600E              BRA.S   SEND_HAND
1957
1958                        *
1959                        *   Xon/Xoff handshake--send Xon if queue can handle more characters
1960                        *
1961              0000 0948 XON_H           EQU *
1962     00000948 6100 0184         BSR     QUEUE_SPACE             d3 := space left in queue
1963     0000094C B67C 005E         CMP.W   #XON_SIZE,D3
1964     00000950 6D40              BLT.S   READ_END                space not big enough to send XON
1965                        *
1966                        *   Send character to indicate card is receiving and set receiving flag
1967                        *
1968     00000952 162A 0041         MOVE.B  XON_CHAR(A2),D3
1969
1970              0000 0956 SEND_HAND       EQU *                   send handshake character (in D3)
1971     00000956 137C 0001         MOVE.B  #1,INTR_EN(A1)          only have receive interrupt enabled
                  0013
1972     0000095C 6100 0036         BSR     SEND                    send char which is in d2
1973     00000960 6606              BNE.S   RESTORE
1974     00000962 157C 0001         MOVE.B  #1,RECEIVING(A2)        turn receiving back on
                  003C
1975              0000 0968 RESTORE         EQU *                   recalculate interrupt enable mask
1976     00000968 1E29 0003         MOVE.B  INTR_SW(A1),D7          save interrupt status
1977     0000096C 4229 0003         CLR.B   INTR_SW(A1)             critical section
1978     00000970 4AAA 0028         TST.L   BUFO_OFF(A2)
1979     00000974 6718              BEQ.S   END_RESTORE
1980     00000976 4A2A 003D         TST.B   XMITTING(A2)
1981     0000097A 6712              BEQ.S   END_RESTORE
1982     0000097C 08E9 0001         BSET    #1,INTR_EN(A1)
                  0013
1983     00000982 4A2A 003B         TST.B   MODEM_ON(A2)
1984     00000986 6706              BEQ.S   END_RESTORE
1985     00000988 08E9 0003         BSET    #3,INTR_EN(A1)
                  0013
1986              0000 098E END_RESTORE     EQU *
1987     0000098E 1347 0003         MOVE.B  D7,INTR_SW(A1)          end critical section
1988
1989              0000 0992 READ_END        EQU *
1990     00000992 4E75              RTS
```



```
1992                        *-------------------------------------------------------------------------------
1993                        *       SEND
1994                        *       ON ENTRY:  d3.B -- character to be sent
1995                        *       ON EXIT :  IF character sent
1996                        *                      THEN Z=1
1997                        *                      ELSE Z=0, S_ERROR updated to newest error
1998                        *       USES    : a4 -- parameter to WAIT
1999                        *                 d7 -- temporary
2000                        *                 d6 -- by called routines
2001                        *-------------------------------------------------------------------------------
2002
2003              0000 0994 SEND            EQU *
2004     00000994 08E9 0001         BSET    #1,MODEM_CONT(A1)       set RTS
                  0019
2005                        *
2006                        *   Wait (with timeout) for transmit registers empty
2007                        *
2008              0000 099A LOOP_THRE       EQU *
2009     0000099A 1E29 001B         MOVE.B  LINE_STAT(A1),D7
2010     0000099E 8F2A 003F         OR.B    D7,S_LINE(A2)           save line status for user
2011     000009A2 CE3C 0020         AND.B   #$20,D7                 look at THRE bit
2012     000009A6 67F2              BEQ.S   LOOP_THRE
2013
2014     000009A8 4A2A 003B         TST.B   MODEM_ON(A2)            skip modem stuff if modem handshake off
2015     000009AC 6722              BEQ.S   XMIT_CHAR
2016                        *
2017                        *   Modem checking depends on if this routine was called from an ISR
2018                        *
2019     000009AE 4A2A 0038         TST.B   IN_ISR(A2)
2020     000009B2 6710              BEQ.S   NOT_ISR
2021
2022     000009B4 6100 0098         BSR     CHECK_DSR_CTS
2023     000009B8 6716              BEQ.S   XMIT_CHAR               modem lines are up, goto transmit
2024
2025     000009BA 257C 0000         MOVE.L  #316,S_ERROR(A2)        CTS false error
                  013C 0034
2026     000009C2 4E75              RTS                             side effect -- Z:=0
2027                        *
2028                        *   Wait (with timeout) for DSR and CTS
2029                        *
2030              0000 09C4 NOT_ISR         EQU *
2031     000009C4 49FA 0088         LEA     CHECK_DSR_CTS,A4        \ call WAIT with appropriate
2032     000009C8 6100 000E         BSR     WAIT                    / function parameter
2033     000009CC 6702              BEQ.S   XMIT_CHAR               no errors, goto transmit
2034     000009CE 4E75              RTS                             (Z=0 still)
2035                        *
2036                        *   Send the character (in d3)
2037                        *
2038              0000 09D0 XMIT_CHAR       EQU *
2039     000009D0 1343 0011         MOVE.B  D3,DATA(A1)             do actual transmit
2040     000009D4 4207              CLR.B   D7                      indicate no errors  (Z := 1)
2041     000009D6 4E75              RTS
```

```
2043                    *-------------------------------------------------------------------------------
2044                    *       WAIT
2045                    *          this function waits with timeout for a condition to happen,
2046                    *          if the condition does not happen within the timeout, then
2047                    *          S_ERROR(A2) is marked with the timeout error.
2048                    *       ON ENTRY:  A4.L  points to the routine which will determine if
2049                    *                        the condition is met.  This routine should have
2050                    *                        the following conditions:
2051                    *                          --uses at the most d7,d6
2052                    *                          --returns Z=1 if the condition is met
2053                    *                          --all routines should have similar timing
2054                    *       ON EXIT:   IF error is found
2055                    *                     THEN Z=0, S_ERROR indicates the error
2056                    *                     ELSE Z=1
2057                    *       USES    :  D4.L  --  timeout counter
2058                    *                  D7,D6 --  can be used by called routine (see above)
2059                    *-------------------------------------------------------------------------------
2060
2061             0000 09D8 WAIT          EQU *
2062    00000 9D8 4E94              JSR     (A4)                    check the condition
2063    00000 9DA 673E              BEQ.S   EXIT_WAIT               exit if condition met (Z=1)
2064
2065    00000 9DC 282A 002E         MOVE.L  TIMEOUT(A2),D4
2066    00000 9E0 672E              BEQ.S   WAIT_LOOP2              infinite timeout if value is 0.
2067
2068    00000 9E2 0838 0001         BTST    #TIMER_PRESENT,SYSFLAG2  SEE IF TIMER EXISTS     ttt JS 8/3/83
                  FEDA
2069    00000 9E8 6732              BEQ.S   WAIT_TMR                 IF SO GO USE IT         ttt JS 8/3/83
2070
2071    00000 9EA 2E04              MOVE.L  D4,D7                   \
2072    00000 9EC E38F              LSL.L   #1,D7                    \    initialize counter
2073    00000 9EE E58C              LSL.L   #2,D4                    \      (multiply by 54)
2074    00000 9F0 D887              ADD.L   D7,D4                     |
2075    00000 9F2 2E04              MOVE.L  D4,D7                     /
2076    00000 9F4 E78C              LSL.L   #3,D4                    /
2077    00000 9F6 D887              ADD.L   D7,D4                   /
2078
2079             0000 09F8 WAIT_LOOP    EQU *
2080    00000 9F8 4AAA 0034         TST.L   S_ERROR(A2)          check for errors saved during wait
2081    00000 9FC 661C              BNE.S   EXIT_WAIT               exit (Z=0)
2082    00000 9FE 4E94              JSR     (A4)                    check the condition
2083    00000A0C 6718               BEQ.S   EXIT_WAIT               exit if conditon met (Z=1)
2084
2085    00000A02 5384               SUBQ.L  #1,D4                   counter := counter - 1
2086    00000A04 6AF2               BPL.S   WAIT_LOOP
2087
2088    00000A06 257C 0000          MOVE.L  #TMO_ERR,S_ERROR(A2) save the timeout error (Z:=0)
                  0011 0034
2089    00000A0E 4E75               RTS
2090
2091             0000 0A10 WAIT_LOOP2   EQU *                    wait loop for infinite timeout
2092    00000A10 4AAA 0034          TST.L   S_ERROR(A2)          check for errors saved by ISRs
2093    00000A14 6604               BNE.S   EXIT_WAIT               exit (Z=0)
2094    00000A16 4E94               JSR     (A4)
2095    00000A18 66F6               BNE.S   WAIT_LOOP2
2096
2097             0000 0A1A EXIT_WAIT    EQU *
```



```
2098    00000A1A 4E75               RTS
2099
2100             0000 0A1C WAIT_TMR     EQU *
2101    00000A1C 1F3C 0001          MOVE.B  #1,-(SP)            SET UP TIMER RECORD  ttt JS 8/3/83
2102    00000A20 2F04               MOVE.L  D4,-(SP)            D4 HAS MS TO WAIT    ttt JS 8/3/83
2103             0000 0A22 WAIT_TMR1    EQU *                                        ttt JS 8/3/83
2104    00000A22 4AAA 0034          TST.L   S_ERROR(A2)         CHECK FOR ERRORS     ttt JS 8/3/83
2105    00000A26 6614               BNE.S   WAIT_TEXIT          BR IF ERROR          ttt JS 8/3/83
2106    00000A28 4E94               JSR     (A4)                CHECK CONDITION      ttt JS 8/3/83
2107    00000A2A 6710               BEQ.S   WAIT_TEXIT           BR IF CONDITION MET  ttt JS 8/3/83
2108    00000A2C 4857               PEA     (SP)                POINT TO TIMER REC   ttt JS 8/3/83
2109    00000A2E 4EB9 0000          JSR     CHECK_TIMER         AND CHECK TIMER      ttt JS 8/3/83
                  0000
2110    00000A34 6AEC               BPL     WAIT_TMR1           IF NO TIMEOUT BRANCH ttt JS 8/3/83
2111    00000A36 5C4F               ADDQ    #6,SP               TIMEOUT, BUT GET ONE ttt JS 5/3/84
2112    00000A38 7814               MOVEQ   #20,D4              MORE CHANCE WITH     ttt JS 5/3/84
2113    00000A3A 60BC               BRA     WAIT_LOOP           SHORT COUNT          ttt JS 5/3/84
2114             0000 0A3C WAIT_TEXIT   EQU *                                        ttt JS 8/3/83
2115    00000A3C 5C4F               ADDQ    #6,SP               CLEANUP STACK        ttt JS 8/3/83
2116    00000A3E 4E75               RTS                         AND DONE!            ttt JS 8/3/83
```

```
                        ********************************************************************************
                        *
                        *       CHECK_DSR_CTS, CHECK_QUEUE
                        *
                        *       FUNCTIONs to be used with WAIT, they all return Z=1 when the
                        *          condition is met.
                        *
                        *       USES:  the function is allowed to use only d6 and d7
                        *
                        ********************************************************************************

                        *
                        *   condition:  queue is empty
                        *
             0000 0A40  CHECK_QUEUE     EQU *
00000A40 3E2A 004C      CHECK_QUEUE     MOVE.W  Q_OUT(A2),D7            (12)
00000A44 BE6A 004A      CHECK_QUEUE     CMP.W   Q_IN(A2),D7             (12)  Z=1 if empty
00000A48 0A3C 0004      CHECK_QUEUE     EORI    #$04,CCR                (20)  invert the Z bit (Z=1 if full)
00000A4C 4E75           CHECK_QUEUE     RTS                             (----> 44 )

                        *
                        *   condition:  DSR=1 and CTS=1  (ok to send with modem handshake)
                        *
             0000 0A4E  CHECK_DSR_CTS   EQU *
00000A4E 1E29 001D      CHECK_DSR_CTS   MOVE.B  MODEM_STAT(A1),D7        (12)
00000A52 8F2A 003E      CHECK_DSR_CTS   OR.B    D7,S_MODEM(A2)          (16)  save modem status for user
00000A56 4607           CHECK_DSR_CTS   NOT.B   D7                      ( 4)
00000A58 CE7C 0030      CHECK_DSR_CTS   AND     #$30,D7                 ( 8)  if both DSR and CTS were true, Z=0
00000A5C 4E71           CHECK_DSR_CTS   NOP                             ( 4)
00000A5E 4E75           CHECK_DSR_CTS   RTS                             (----> 44 )
```



```
                        ********************************************************************************
                        *
                        *       Buffer routines
                        *
                        ********************************************************************************

                        *-------------------------------------------------------------------------------
                        *       INIT_QUEUE
                        *          initializes the queue descriptor.
                        *-------------------------------------------------------------------------------

             0000 0A60  INIT_QUEUE      EQU *
00000A60 357C 0086      INIT_QUEUE      MOVE.W  #BUFFER_SIZE,Q_SIZE(A2)         * initialize  queue_size
             0048
00000A66 426A 004A      INIT_QUEUE      CLR.W   Q_IN(A2)                        *  queue_in := 0
00000A6A 426A 004C      INIT_QUEUE      CLR.W   Q_OUT(A2)                       *  queue_out := 0
00000A6E 4E75           INIT_QUEUE      RTS

                        *-------------------------------------------------------------------------------
                        *       QUEUE_EMPTY
                        *          tells if queue is empty.
                        *          ON EXIT:  Z=empty (IF EMPTY THEN Z:=1 ELSE Z:=0)
                        *          USES   :  D7
                        *-------------------------------------------------------------------------------

             0000 0A70  QUEUE_EMPTY     EQU *
00000A70 3E2A 004C      QUEUE_EMPTY     MOVE.W  Q_OUT(A2),D7
00000A74 BE6A 004A      QUEUE_EMPTY     CMP.W   Q_IN(A2),D7             *  RETURN( queue_in = queue_out )
00000A78 4E75           QUEUE_EMPTY     RTS

                        *-------------------------------------------------------------------------------
                        *       QUEUE_FULL
                        *          tells if queue is full.
                        *          ON EXIT:  Z=full (IF FULL THEN Z:=1 ELSE Z:=0)
                        *          USES   :  D7
                        *-------------------------------------------------------------------------------

             0000 0A7A  QUEUE_FULL      EQU *
00000A7A 3E2A 004A      QUEUE_FULL      MOVE.W  Q_IN(A2),D7             *
00000A7E 5247           QUEUE_FULL      ADDQ.W  #1,D7                   *
00000A80 BE6A 004C      QUEUE_FULL      CMP.W   Q_OUT(A2),D7            *  queue_out = queue_in+1 ?
00000A84 6602           QUEUE_FULL      BNE.S   CHECK_OR                *
00000A86 4E75           QUEUE_FULL      RTS                             *        ( YES so return with Z=1)
             0000 0A88  CHECK_OR        EQU *
00000A88 BE6A 0048      CHECK_OR        CMP.W   Q_SIZE(A2),D7           *  queue_in+1 = queue_size ?
00000A8C 6702           CHECK_OR        BEQ.S   CHECK_AND               *
00000A8E 4E75           CHECK_OR        RTS                             *        ( NO  so return with Z=0)
             0000 0A90  CHECK_AND       EQU *
00000A90 3E2A 004C      CHECK_AND       MOVE.W  Q_OUT(A2),D7            *  queue_out = 0  ?
00000A94 4E75           CHECK_AND       RTS                             *        ( ANSWER is result of function )
```

```
                        *-------------------------------------------------------------------------------
                        *       INQUEUE
                        *          puts a character in the queue
                        *          ON ENTRY:  d2.B - character to be put in the queue
                        *                     buffer NOT full !!
                        *          USES    :  a4.L - queue_addr
                        *                     d7.W - queue_in
                        *-------------------------------------------------------------------------------

            0000 0A96 INQUEUE       EQU *
00000A96 3E2A 004A          MOVE.W  Q_IN(A2),D7             *
00000A9A 1582 704E          MOVE.B  D2,Q_BUFFER(A2,D7.W)    *  (queue_addr+queue_in)^ := char

00000A9E 5247               ADDQ.W  #1,D7                   *  queue_in := queue_in+1

00000AA0 BE6A 0048          CMP.W   Q_SIZE(A2),D7           *
00000AA4 6C06               BGE.S   RESET_IN                *  IF queue_in >= queue_size
00000AA6 3547 004A          MOVE.W  D7,Q_IN(A2)             *
00000AAA 4E75               RTS                             *
            0000 0AAC RESET_IN      EQU *                   *
00000AAC 426A 004A          CLR.W   Q_IN(A2)                *    THEN queue_in := 0
00000AB0 4E75               RTS                             *

                        *-------------------------------------------------------------------------------
                        *       OUTQUEUE
                        *          take the next character out of the queue
                        *          ON ENTRY:  buffer NOT full
                        *          ON_EXIT :  d2.B - character from the queue
                        *          USES    :  a4.L - queue_addr
                        *                     d7.W - queue_in
                        *-------------------------------------------------------------------------------

            0000 0AB2 OUTQUEUE      EQU *
00000AB2 3E2A 004C          MOVE.W  Q_OUT(A2),D7            *
00000AB6 1432 704E          MOVE.B  Q_BUFFER(A2,D7.W),D2    *  char := (queue_addr+queue_out)^

00000ABA 5247               ADDQ.W  #1,D7                   *  queue_out := queue_out+1

00000ABC BE6A 0048          CMP.W   Q_SIZE(A2),D7           **
00000AC0 6C06               BGE.S   RESET_OUT               *  IF queue_out >= queue_size
00000AC2 3547 004C          MOVE.W  D7,Q_OUT(A2)            *
00000AC6 4E75               RTS                             *
            0000 0AC8 RESET_OUT     EQU *                   *
00000AC8 426A 004C          CLR.W   Q_OUT(A2)               *    THEN queue_out := 0
00000ACC 4E75               RTS                             **
```



```
                        *-------------------------------------------------------------------------------
                        *       QUEUE_SPACE
                        *          returns amount of space remaining in the queue
                        *          ON EXIT:  d3.W - contains the space remaining in the queue.
                        *-------------------------------------------------------------------------------

            0000 0ACE QUEUE_SPACE   EQU *
00000ACE 362A 004C          MOVE.W  Q_OUT(A2),D3            *
00000AD2 966A 004A          SUB.W   Q_IN(A2),D3             *  IF  queue_in >= queue_out
00000AD6 6E08               BGT.S   OUT_GREATER             *
                                                            *
00000AD8 5343               SUBQ.W  #1,D3                   *  THEN queue_space :=
00000ADA D66A 0048          ADD.W   Q_SIZE(A2),D3           *     queue_size + queue_out-queue_in - 1
00000ADE 4E75               RTS
            0000 0AE0 OUT_GREATER   EQU *
00000AE0 5343               SUBQ.W  #1,D3                   *  ELSE queue_space :=
00000AE2 4E75               RTS                             *     queue_out-queue_in - 1
                            END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

*** 68000 ASSEMBLER SYMBOL TABLE DUMP ***

EXTERNAL SYMBOLS

SYMBOL	TYPE	DEF	VALUE
ABORT_IO	ABS	113	00000002
CHECK_TIMER	ABS	114	0000000A
DELAY_TIMER	ABS	114	00000007
IODECLARATIONS	ABS	253	0000000D
LOGEOT	ABS	113	00000005
SYSGLOBALS	ABS	254	00000011

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	1297		00000520
ACK	ABS	511		00000006
ACK_CHAR	ABS	487		00000044
ACK_SIZE	ABS	514		00000050
ASM_INIT	REL	541		0000000C
AVAIL_OFF	ABS	193		00000034
BADTMO	ABS	271		0000000B
BAD_RDS	ABS	279		00000013
BAD_SCT	ABS	280		00000014
BAUD	REL	620		000000BC
BAUD_SW	ABS	447		00000005
BUFFER_SIZE	ABS	507		00000086
BUFI_OFF	ABS	184		00000024
BUFO_OFF	ABS	185		00000028
BUF_BUSY	ABS	269		00000009
BUSY_ERR	REL	1730		00000828
CALC_DIV	REL	1123		000003EA
CCR	STREG	0		00000005
CHECK_AND	REL	2197		00000A90
CHECK_BREAK	REL	1385		000005EE
CHECK_DSR_CTS	REL	2141		00000A4E
CHECK_ERROR	REL	1845		000008D8
CHECK_FHS	REL	1623		00000790
CHECK_OR	REL	2193		00000A88
CHECK_QUEUE	REL	2132		00000A40
CHECK_XFER_IN	REL	1720		00000818
CHECK_XFER_OUT	REL	1725		00000820
CHECK_XIN	REL	1493		000006C4
CHECK_XOFF	REL	1429		0000064A
CLEAR_CUT	REL	1694		000007FA
CLEAR_XFER	REL	1687		000007E4
CLR_XOUT	REL	1357		000005B6
CONNECT	REL	1782		0000085A

CONNECTED	ABS	477		0000003A
CONT_0	REL	1084		000003AA
CONT_1	REL	1089		000003B2
CONT_12	REL	1182		00000472
CONT_13	REL	1191		00000486
CONT_14	REL	1199		00000496
CONT_15	REL	1206		000004A0
CONT_16	REL	1210		000004A6
CONT_17	REL	1214		000004AC
CONT_18	REL	1218		000004B2
CONT_19	REL	1222		000004B8
CONT_20	REL	1226		000004BE
CONT_3	REL	1108		000003DC
CONT_4	REL	1142		00000418
CONT_5	REL	1163		0000044C
CONT_6	REL	1167		00000452
CONT_7	REL	1173		0000045C
CONT_7R	REL	1180		00000470
CONT_ERROR	REL	1080		000003A4
CONT_TABLE	REL	1057		0000037A
CONV_CHAR	ABS	488		00000045
CRD_DWN	ABS	281		00000015
C_ADR	ABS	183		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	454		00000011
DC1	ABS	508		00000011
DC3	ABS	509		00000013
DFC	STREG	0		00000008
DISCONNECT	REL	1800		0000087E
DIV0	ABS	456		00000011
DIV1	ABS	457		00000013
DMA_ERR	REL	1556		00000716
DONT_SET	REL	961		000002EC
DO_BIT_4	REL	834		000001E2
DO_BIT_5	REL	839		000001EC
DUMP_BUFFER	REL	1652		000007A8
DUMP_LOOP	REL	1656		000007AE
EIRB_OFF	ABS	186		0000002C
END_ERR	REL	1555		00000716
END_ISR	REL	1500		000006D2
END_RESTORE	REL	1986		0000098E
END_STS_2	REL	844		000001F6
END_XIN	REL	1666		000007C6
END_XOUT	REL	1339		00000580
ENQ	ABS	510		00000005
ENQ_CHAR	ABS	486		00000043
ENQ_H	REL	1948		00000938
ENQ_HAND	REL	1457		0000067E
EOD_SEEN	ABS	282		00000016
ERROR_EXIST	REL	1849		000008E0

```
ERROR_INTR      REL     1285                            00000502
ESC_CODE        ABS      290 SYSGLOBALS      +          FFFFFFFE
EXIT_14         REL     1204                            0000049E
EXIT_C1         REL     1108                            000003DA
EXIT_C4         REL     1161                            0000044A
EXIT_CONNECT    REL     1791                            0000087C
EXIT_DUMP       REL     1675                            000007DC
EXIT_TFR        REL     1631                            000007A6
EXIT_WAIT       REL     2097                            00000A1A
GET_CHAR        REL     1920                            00000918
HAND_TABLE      REL     1407                            00000616
H_ISR_PM        ABS      182                            0000001C
H_ISR_PR        ABS      180                            00000014
H_ISR_SL        ABS      181                            00000018
H_TABLE         REL     1939                            00000930
ID_REG          ABS      445                            00000001
IGNORE_PE       ABS      489                            00000046
INITALL         REL      591                            0000007E
INIT_QUEUE      REL     2162                            00000A60
INPUT_END       REL     1483                            000006B6
INPUT_INTR      REL     1367                            000005CA
INPUT_XFER      REL     1586                            00000744
INQUEUE         REL     2210                            00000A96
INTR_EN         ABS      455                            00000013
INTR_EXIST      REL     1267                            000004EA
INTR_ID         ABS      458                            00000015
INTR_SW         ABS      446                            00000003
INTR_TABLE      REL     1274                            000004FA
INTR_XFER       REL     1576                            00000738
IN_ISR          ABS      475                            00000038
IOESCAPE        REL     1745                            0000082E
IOE_ERROR       ABS      285                            FFFFFFE6
IOE_RSLT        ABS      287 IODECLARATIONS  +          FFFFFFBE
IOE_SC          ABS      288 IODECLARATIONS  +          FFFFFFBA
IO_MISC         ABS      283                            00000017
IO_SC           ABS      187                            0000002D
IS21            REL      881                            00000248
IS43            REL      877                            0000023C
IS77            REL      873                            00000230
IS85            REL      869                            00000224
ISR_ENTRY       ABS      178                            00000000
LINE_CONT       ABS      459                            00000017
LINE_STAT       ABS      461                            0000001B
LINE_SW         ABS      448                            00000007
LOOP_LAST       REL     1345                            00000592
LOOP_THRE       REL     2008                            0000099A
MA              ABS      192                            00000033
MA_W            ABS      191                            00000032
MODEM_CONT      ABS      460                            00000019
MODEM_INTR      REL     1291                            00000514
MODEM_ON        ABS      478                            0000003B
MODEM_STAT      ABS      462                            0000001D
MOVE_OUT        REL     1327                            00000562
NOT_APIB        ABS      262                            00000002
NOT_ISR         REL     2030                            000009C4
NOT_LSTN        ABS      276                            00000010
NOT_TALK        ABS      275                            0000000F
```



```
NO_ACTL         ABS      263                            00000003
NO_CARD         ABS      261                            00000001
NO_CONVERT      REL     1398                            00000608
NO_DATA         ABS      266                            00000006
NO_DMA          ABS      273                            0000000D
NO_DRV          ABS      272                            0000000C
NO_DVC          ABS      264                            00000004
NO_HAND         REL     1477                            000006AC
NO_SCTL         ABS      278                            00000012
NO_SPACE        ABS      265                            00000005
NO_WORD         ABS      274                            0000000E
OK_XMIT         REL     1880                            000008FE
OUTPUT_INTR     REL     1303                            00000532
OUTPUT_XFER     REL     1607                            0000076A
OUTQUEUE        REL     2233                            00000AB2
OUT_GREATER     REL     2261                            00000AE0
OVERRUN         REL     1487                            000006BC
OVERRUN_ERROR   ABS      513                            0000013A
PUTINQ          REL     1478                            000006AC
QUEUE_EMPTY     REL     2175                            00000A70
QUEUE_FULL      REL     2187                            00000A7A
QUEUE_SPACE     REL     2253                            00000ACE
Q_BUFFER        ABS      495                            0000004E
Q_DESCRIPT      ABS      491                            00000048
Q_IN            ABS      493                            0000004A
Q_OUT           ABS      494                            0000004C
Q_SIZE          ABS      492                            00000048
RCVR_BLK        ABS      291 SYSGLOBALS      +          FFFFFFF6
RDIVU           REL     1760                            00000844
READ_END        REL     1989                            00000992
READ_UART       REL      923                            0000029A
RECEIVING       ABS      479                            0000003C
REGULAR         REL      885                            00000254
REG_MAX         ABS      517                            00000014
RESET_IN        REL     2220                            00000AAC
RESET_OUT       REL     2243                            00000AC8
RESET_REG       ABS      444                            00000001
RESTORE         REL     1975                            00000968
ROUND           REL     1770                            00000854
RS_RS           REL      434                            00000000
RS_RS_INIT      REL      526                            00000002
RS_RS_ISR       REL     1244                            000004D4
RS_RS_RDB       REL      644                            000000DC
RS_RS_RDS       REL      758                            00000166
RS_RS_RDW       REL      701                            0000011A
RS_RS_TFR       REL     1511                            000006D8
RS_RS_WTB       REL      673                            000000FC
RS_RS_WTC       REL     1031                            00000359
RS_RS_WTW       REL      728                            00000140
SC_BUSY         ABS      268                            00000008
SEND            REL     2003                            00000994
SEND_ACK        REL     1466                            00000698
SEND_HAND       REL     1970                            00000956
SER_FHS         REL     1570                            0000072A
SETALL          REL      608                            000000A6
SET_BIT_0       REL      831                            000001DE
SET_OUT         REL     1710                            00000812
```

SET_XFER	REL	1704	00000800
SFC	STREG	0	00000009
SOFT_RESET	REL	1815	00000896
SP	AREG	0	00000007
SR	STREG	0	00000006
STS_0	REL	813	000001BC
STS_1	REL	817	000001C2
STS_10	REL	946	000002C6
STS_11	REL	965	000002F0
STS_12	REL	979	00000310
STS_13	REL	983	00000316
STS_14	REL	987	0000031C
STS_15	REL	998	00000334
STS_16	REL	1002	0000033A
STS_17	REL	1006	00000340
STS_18	REL	1010	00000346
STS_19	REL	1014	0000034C
STS_2	REL	821	000001C8
STS_20	REL	1018	00000352
STS_3	REL	848	000001FA
STS_4	REL	894	00000262
STS_5	REL	905	00000276
STS_6	REL	909	0000027C
STS_7	REL	927	000002A0
STS_7B	REL	935	000002B6
STS_8	REL	938	000002BA
STS_9	REL	942	000002C0
STS_ERROR	REL	809	000001B6
STS_TABLE	REL	786	0000018C
SYSFLAG2	ABS	294	FFFFFEDA
S_ERROR	ABS	474	00000034
S_HANDSH	ABS	483	00000040
S_LINE	ABS	482	0000003F
S_MODEM	ABS	481	0000003E
TACT_OFF	ABS	208	00000007
TBSZ_OFF	ABS	231	00000018
TBUF_OFF	ABS	230	00000014
TCHR_OFF	ABS	227	0000000E
TCNTERR	ABS	270	0000000A
TCNT_OFF	ABS	229	00000010
TDIR_OFF	ABS	225	0000000D
TEMP_OFF	ABS	232	0000001C
TEMP_SIZE	ABS	506	000000A0
TEND_OFF	ABS	223	0000000B
TERM_HAND	REL	1440	0000065E
TFIL_OFF	ABS	233	00000020
TFR_ERR	ABS	267	00000007
TIMEOUT	ABS	188	0000002E
TIMER_PRESENT	ABS	293	00000001
TMO_ERR	ABS	277	00000011
TTMP_OFF	ABS	206	00000000
TT_BURST	ABS	244	00000003
TT_DMA	ABS	243	00000002
TT_FHS	ABS	245	00000004
TT_INT	ABS	242	00000001
TUSR_OFF	ABS	209	00000009
T_BW_OFF	ABS	221	0000000A

T_DMAPRI	ABS	238	00000030
T_PM_OFF	ABS	237	0000002C
T_PR_OFF	ABS	234	00000024
T_SC_OFF	ABS	207	00000005
T_SL_OFF	ABS	236	00000028
UNDERSCORE	ABS	512	0000005F
USER_ISR	ABS	179	00000014
USP	STREG	0	00000007
VAL_ERR	REL	1187	00000480
VAL_ERR2	REL	1233	000004CE
VBR	STREG	0	0000000A
WAIT	REL	2061	000009D8
WAIT_BREAK	REL	1097	000003C2
WAIT_BREAK2	REL	1104	000003D4
WAIT_FHS	REL	1628	0000079E
WAIT_GET	REL	1895	00000906
WAIT_LOOP	REL	2079	000009F8
WAIT_LOOP2	REL	2091	00000A10
WAIT_SEND	REL	1869	000008F8
WAIT_TEXIT	REL	2114	00000A3C
WAIT_TMR	REL	2100	00000A1C
WAIT_TMR1	REL	2103	00000A22
WORD_ERR	REL	1562	00000720
XFER_ERR	REL	1557	00000716
XFER_OUT	REL	1315	00000542
XFER_TABLE	REL	1539	00000702
XIN_ACT	ABS	476	00000039
XMITTING	ABS	480	0000003D
XMIT_CHAR	REL	2038	000009D0
XOFF_CHAR	ABS	485	00000042
XOFF_SIZE	ABS	515	00000028
XON_CHAR	ABS	484	00000041
XON_H	REL	1961	00000948
XON_HAND	REL	1413	0000061E
XON_SIZE	ABS	516	0000005E

Description

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 [3.0] 12/26/84 21:44:43  ASSEMBLY OF RSTRINT.TEXT                                  *** File name: RSTRINT ***

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 [3.0] 12/26/84 21:44:43  ASSEMBLY OF RSTRINT.TEXT                                  *** File name: RSTRINT ***

   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
```

Description

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 [3.0] 12/26/84 21:44:58  ASSEMBLY OF SCAN.TEXT                                   *** File name: SCAN ***

PASS 1 COMPLETE, ERRORS: 0
    1                       ************************************************************************
    2                       *
    3                       *       SCAN function -- equivalent to UCSD SCAN
    4                       *         --written for M68000 by ------------, 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 [3.0] 12/26/84 21:44:58  ASSEMBLY OF SCAN.TEXT                                   *** File name: SCAN ***

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
DFC	STREG	0		00000008
EXIT	REL	55		0000004A
SC@N1	REL	29		00000016
SC@N2	REL	35		00000022
SC@N3	REL	44		00000034
SC@N4	REL	50		00000040
SFC	STREG	0		00000009
SP	AREG	0		00000007
SR	STREG	0		00000006
USP	STREG	0		00000007
VBR	STREG	0		0000000A

SETSTUFF

Description

SETSTUFF contains assembly language to perform set assignment and adding an element to a set.

Usage

The Compiler emits calls to these routines.

```
PAGE   1 [3.0] 12/26/84 21:45:04  ASSEMBLY OF SETSTUFF.TEXT                                  *** File name: SETSTUFF ***

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
```

```
PAGE   2 [3.0] 12/26/84 21:45:04  ASSEMBLY OF SETSTUFF.TEXT                                  *** File name: SETSTUFF ***

   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)
```

```
PAGE   3 [3.0] 12/26/84 21:45:04  ASSEMBLY OF SETSTUFF.TEXT                                  *** File name: SETSTUFF ***

   71                          *---------------------------------------
   72                                    DEF asm_aDELEMENT
   73                          *---------------------------------------
   74                 0000 007E asm_aDELEMENT EQU         *
   75      0000007E 205F                      movea.l          (sp)+,a0          return address
   76      00000080 201F                      move.l           (sp)+,d0          element number to add to set
   77      00000082 0C80 0000                 cmpi.l           #255,d0
                    00FF
   78      00000088 6302                      bls.s            strt
   79      0000008A 4E47                          trap    #7
   80      0000008C 225F              strt    movea.l          (sp)+,a1          source address
   81      0000008E 2457                      movea.l          (sp),a2           destination address
   82      00000090 3E19                      move.w           (a1)+,d7          get set size of source
   83      00000092 34C7                      move.w           d7,(a2)+          store size value
   84      00000094 B3CA                      cmpa.l           a2,a1             see if source and destination are equal
   85      00000096 670C                      beq.s            insert
   86                          * copy source set to the destination set
   87      00000098 264A              setcopy movea.l          a2,a3             save destination address
   88      0000009A 3C07                      move.w           d7,d6             save size for destination
   89      0000009C 6F06                      ble.s            insert            check for size of zero
   90      0000009E 36D9              rept    move.w           (a1)+,(a3)+       sets are always an even number of bytes
   91      000000A0 5546                      subq.w           #2,d6
   92      000000A2 6EFA                      bgt.s            rept
   93                          * insert an element in a set, adjusting the size of the destination if needed
   94      000000A4 48C0              insert  ext.l            d0
   95      000000A6 81FC 0010                 divs             #16,d0             byte offset in low word
   96      000000AA 2A00                      move.l           d0,d5
   97      000000AC 4845                      swap             d5                bit offset from left of byte
   98      000000AE 9A7C 000F                 sub.w            #15,d5
   99      000000B2 4445                      neg.w            d5                bit offset from right
  100      000000B4 E340                      asl              #1,d0             make d0 a byte offset
  101      000000B6 3200                      move.w           d0,d1             compute final size into d1
  102      000000B8 5441                      addq.w           #2,d1             put zeros in the two bytes containing
  103      000000BA 3401                      move.w           d1,d2             the new bit if it is beyond current size
  104      000000BC 926A FFFE                 sub.w            -2(a2),d1
  105      000000C0 6F0E                      ble.s            exxiit
  106      000000C2 3542 FFFE                 move.w           d2,-2(a2)         store appropriate size for set
  107      000000C6 47F2 2000                 lea              0(a2,d2),a3
  108      000000CA 4263              zerout  clr.w            -(a3)
  109      000000CC 5541                      subq.w           #2,d1
  110      000000CE 6EFA                      bgt.s            zerout
  111      000000D0 0885 0003 exxiit  bclr             #3,d5             { received upgrade 9/9 }
  112      000000D4 6706                      beq.s            skiipp
  113      000000D6 0BF2 0000                 bset             d5,0(a2,d0)
  114      000000DA 4ED0                      jmp              (a0)
  115      000000DC 0BF2 0001 skiipp  bset             d5,1(a2,d0)
  116      000000E0 4ED0                      jmp              (a0)
  117                          *---------------------------------------
  118                                 end
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

```
PAGE   4 [3.0] 12/26/84 21:45:04  ASSEMBLY OF SETSTUFF.TEXT                                  *** File name: SETSTUFF ***

 *** 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
DFC                 STREG     0                        00000008
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
SFC                 STREG     0                        00000009
SKIIPP              REL     115                        000000DC
SP                  AREG      0                        00000007
SR                  STREG     0                        00000006
STRT                REL      80                        0000008C
TESTHIGH            REL      40                        00000040
USP                 STREG     0                        00000007
VBR                 STREG     0                        0000000A
ZEROUT              REL     108                        000000CA
```

Description

STRG1 contains assembly language routines to support the standard Pascal functions:

- STRLTRIM
- STRRTRIM
- STRRPT
- STRMOVE

Usage

The Compiler emits calls to these routines.

```
PAGE   1 [3.0] 12/26/84 21:45:12  ASSEMBLY OF STRG1.TEXT                                   *** File name: STRG1 ***

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 [3.0] 12/26/84 21:45:12  ASSEMBLY OF STRG1.TEXT                                   *** File name: STRG1 ***

   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 checkirg.
                        *
 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   retrrn

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
DFC	STREG	0		00000008
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	238		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
SFC	STREG	0	00000009
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
VBR	STREG	0	0000000A

Description

STRG2 contains assembly language routines to support the standard Pascal procedures:

- STRAPPEND
- STRINSERT
- STRDELETE
- APPEND
- INSERT
- DELETE
- COPY
- CONCAT

Usage

The Compiler emits calls to these routines.

```
PAGE   1 [3.0] 12/26/84 21:44:50  ASSEMBLY OF STRG2.TEXT                                   *** File name: STRG2 ***

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   retrnn
   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 6E58                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        retrnn  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 [3.0] 12/26/84 21:44:50  ASSEMBLY OF STRG2.TEXT                                   *** File name: STRG2 ***

   59    0000002C 2A1F                move.l  (sp)+,INDEX
   60    0000002E 6F44                ble.s   error
   61    00000030 245F                movea.l (sp)+,DADDR
   62    00000032 7800                moveq   #0,DMAX
   63    00000034 181F                move.b  (sp)+,DMAX
   64    00000036 225F                movea.l (sp)+,SADDR
   65    00000038 7C00                moveq   #0,TEMP1
   66    0000003A 1C12                move.b  (DADDR),TEMP1
   67    0000003C 7000                moveq   #0,SLEN
   68    0000003E 1019                move.b  (SADDR)+,SLEN
   69    00000040 6730                beq.s   rettrn
   70
   71    00000042 2E06                move.l  TEMP1,TEMP2
   72    00000044 5287                addq.l  #1,TEMP2
   73    00000046 BE85                cmp.l   INDEX,TEMP2
   74    00000048 670A                beq.s   apend0
   75    0000004A 6D28                blt.s   error
   76    0000004C 3E06                move.w  TEMP1,TEMP2
   77    0000004E DE40                add.w   SLEN,TEMP2
   78    00000050 BE44                cmp.w   DMAX,TEMP2
   79    00000052 6E20                bgt.s   error
   80    00000054 14C7                move.b  TEMP2,(DADDR)+
   81    00000056 45F2 6000           lea     0(DADDR,TEMP1.W),DADDR
   82    0000005A 49F2 0000           lea     0(DADDR,SLEN.W),ATEMP
   83    0000005E 3E06                move.w  TEMP1,TEMP2
   84    00000060 9E45                sub.w   INDEX,TEMP2
   85    00000062 1922        insert1 move.b  -(DADDR),-(ATEMP)
   86    00000064 5347                subq.w  #1,TEMP2
   87    00000066 6CFA                bge.s   insert1
   88    00000068 43F1 0000           lea     0(SADDR,SLEN.W),SADDR
   89    0000006C 1921        insert2 move.b  -(SADDR),-(ATEMP)
   90    0000006E 5340                subq.w  #1,SLEN
   91    00000070 6EFA                bgt.s   insert2
   92    00000072 4ED0        rettrn  jmp     (RET)
   93                         *
   94                         *
   95    00000074 3B7C FFF8 error         move.w     #-8,ESCAPECODE(a5)
                  FFFE
   96    0000007A 4E4A                    trap       #10
   97                         *
   98                         *
   99                         *
  100                         *  procedure scopy(var dest: string;
  101                         *                       src: stringmax;
  102                         *                       index,length: integer);
  103                         *
  104                         *
  105               0000 007C ASM_SCOPY     EQU *
  106                                 DEF   ASM_SCOPY
  107                         *
  108    0000007C 205F                movea.l (sp)+,RET
  109    0000007E 201F                move.l  (sp)+,SLEN
  110    00000080 6E0C                bgt.s   copy0
  111    00000082 DFFC 0000                adda.l     #8,sp
                  0008
  112    00000088 245F                     movea.l    (sp)+,DADDR
  113    0000008A 4212                     clr.b      (DADDR)
```

```
PAGE   3 [3.0] 12/26/84 21:44:50  ASSEMBLY OF STRG2.TEXT                                  *** File name: STRG2 ***

  114    0000008C 4ED0                    jmp       (RET)
  115    0000008E 2A1F       copy0   move.l  (sp)+,INDEX
  116    00000090 6FE2               ble.s   error
  117
  118    00000092 225F               movea.l (sp)+,SADDR
  119    00000094 245F               movea.l (sp)+,DADDR
  120    00000096 7400               moveq   #0,CURSLEN
  121    00000098 1411               move.b  (SADDR),CURSLEN
  122    0000009A 5242               addq.w  #1,CURSLEN
  123    0000009C 9485               sub.l   INDEX,CURSLEN
  124    0000009E 6DD4               blt.s   error
  125
  126    000000A0 B042               cmp.w   CURSLEN,SLEN
  127    000000A2 6ED0               bgt.s   error
  128
  129    000000A4 14C0               move.b  SLEN,(DADDR)+
  130    000000A6 43F1 5000          lea     0(SADDR,INDEX.W),SADDR
  131    000000AA 14D9       copy1   move.b  (SADDR)+,(DADDR)+
  132    000000AC 5340               subq.w  #1,SLEN
  133    000000AE 6EFA               bgt.s   copy1
  134    000000B0 4ED0               jmp     (RET)
  135                        *
  136                        *
  137                        *
  138                        *  procedure delete(var dest: string;
  139                        *                   index,lergth: integer);
  140                        *
  141                        *
  142             0000 00B2 ASM_DELETE    EQU *
  143                               DEF   ASM_DELETE
  144                        *
  145    000000B2 205F               movea.l (sp)+,RET
  146    000000B4 201F               move.l  (sp)+,SLEN
  147    000000B6 6E08               bgt.s   del0
  148    000000B8 DFFC 0000               adda.l     #8,sp
                  0008
  149    000000BE 4ED0                    jmp       (RET)
  150    000000C0 2A1F       del0    move.l  (sp)+,INDEX
  151    000000C2 6FB0               ble.s   error
  152
  153    000000C4 245F               movea.l (sp)+,DADDR
  154    000000C6 7400               moveq   #0,CURSLEN
  155    000000C8 1412               move.b  (DADDR),CURSLEN
  156    000000CA 9480               sub.l   SLEN,CURSLEN
  157    000000CC 6DA6               blt.s   error
  158
  159    000000CE 3C02               move.w  CURSLEN,TEMP1
  160    000000D0 5246               addq    #1,TEMP1
  161    000000D2 9C45               sub.w   INDEX,TEMP1
  162    000000D4 6D9E               blt.s   error
  163
  164    000000D6 1482               move.b  CURSLEN,(DADDR)
  165    000000D8 45F2 5000          lea     0(DADDR,INDEX.W),DADDR
  166    000000DC 43F2 0000          lea     0(DADDR,SLEN.W),SADDR
  167    000000E0 14D9       del1    move.b  (SADDR)+,(DADDR)+
  168    000000E2 5346               subq.w  #1,TEMP1
  169    000000E4 6EFA               bgt.s   del1
```

```
PAGE   4 [3.0] 12/26/84 21:44:50  ASSEMBLY OF STRG2.TEXT                                  *** File name: STRG2 ***

  170    000000E6 4ED0               jmp     (RET)
  171
  172                               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	142		000000B2
ASM_INSERT	REL	55		0000002A
ASM_SAPPEND	REL	25		00000000
ASM_SCOPY	REL	105		0000007C
ATEMP	AREG	5		00000004
CCR	STREG	0		00000005
COPY0	REL	115		0000008E
COPY1	REL	131		000000AA
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	150		000000C0
DEL1	REL	167		000000E0
DFC	STREG	0		00000008
DINDEX	DREG	10		00000003
DMAX	DREG	11		00000004
ERROR	REL	95		00000074
ESCAPECODE	ABS	18	SYSGLOBALS +	FFFFFFFE
INDEX	DREG	13		00000005
INSERT1	REL	85		00000062
INSERT2	REL	89		0000006C
RET	AREG	1		00000000
RETRRNN	REL	47		00000028
RETTRN	REL	92		00000072
SADDR	AREG	2		00000001
SFC	STREG	0		00000009
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
VBR	STREG	0	0000000A

Description

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 [3.0] 12/26/84 21:01:01  ASSEMBLY OF STRINT.TEXT                                   *** File name: STRINT ***

PASS 1 COMPLETE. ERRORS: 0
    1                                   def fs_fwritestrint
    2                                   refa sysglobals,M68KTYPE                (rdq)
    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 [3.0] 12/26/84 21:01:01  ASSEMBLY OF STRINT.TEXT                                   *** File name: STRINT ***

   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
                  86A0
   68    00000046 651C                 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 4A38 0000            TST.B   M68KTYPE
   74    0000004E 6604                 BNE.S   DIV0
   75    00000050 40C4                 move    sr,extend               remainder in extend bit
   76    00000052 6002                 BRA.S   DIV1
   77    00000054 42C4       DIV0      DC.W    $42C4                   MOVE    CCR,EXTEND  ??????      (rdq)
   78    00000056 84FC C350  DIV1      divu    #50000,int              divide by 50000
   79    0000005A 3602                 move.w  int,high                quotient is high part
   80    0000005C 4242                 clr.w   int                     zap quotient
   81    0000005E 4842                 swap    int                     get remainder
   82    00000060 44C4                 move    extend,ccr              reconstruct full remainder
   83    00000062 E392                 roxl.l  #1,int                  integer is now mod 100000
   84
   85    00000064 61A8       small     bsr.s   get_digits              extract low order digits
   86
   87    00000066 4A43                 tst.w   high                    check for any high order digits
   88    00000068 6710                 beq.s   finish
   89    0000006A B27C 0005  11        cmp     #5,digits               there should be at least 5 digits
   90    0000006E 6706                 beq.s   12
   91    00000070 1506                 move.b  zero,-(dindex)
   92    00000072 5241                 addq    #1,digits
   93    00000074 60F4                 bra.s   11
   94    00000076 3403       12        move    high,int
   95    00000078 6194                 bsr.s   get_digits              get high order digits
   96
   97               0000 007A finish   equ *
   98
   99    0000007A 2C14                 move.l  (aindex),index
  100    0000007C 6F60                 ble.s   error                   error if index < 1
  101
  102    0000007E 7600                 moveq   #0,strlen
  103    00000080 1611                 move.b  (straddr),strlen        current length of string
  104
  105    00000082 2803                 move.l  strlen,temp
  106    00000084 5284                 addq.l  #1,temp
  107    00000086 BC84                 cmp.l   temp,index
  108    00000088 6E54                 bgt.s   error                   error if index > strlen(s)+1
  109
  110    0000008A 3401                 move    digits,intlen           compute minimum size of new integer
  111    0000008C 4A00                 tst.b   sign
  112    0000008E 6702                 beq.s   nosign
  113    00000090 5242                 addq    #1,intlen               add 1 for minus sign
  114               0000 0092 nosign   equ *
```

```
00000092 4A47              tst     fieldwidth      field width is 12 if passed negative
00000094 6C02              bge.s   nodefault
00000096 7E0C              moveq   #12,fieldwidth
          0000 0098 nodefault equ *

00000098 BE42              cmp   intlen,fieldwidth       fieldwidth is at least size of integer
0000009A 6C02              bge.s   bigger
0000009C 3E02              move    intlen,fieldwidth
          0000 009E bigger  equ *

0000009E 2A06              move.l  index,newindex          compute new index
000000A0 DA47              add     fieldwidth,newindex

000000A2 3805              move    newindex,temp           string length is maximum of
000000A4 5344              subq    #1,temp                         old length, newindex-1
000000A6 B644              cmp     temp,strlen
000000A8 6C02              bge.s   longer
000000AA 3604              move    temp,strlen
          0000 00AC longer  equ     *

000000AC 4244              clr     temp
000000AE 1817              move.b  (sp),temp               retrieve strmax
000000B0 B644              cmp     temp,strlen
000000B2 6E2A              bgt.s   error                   error if new length > strmax(s)

000000B4 1283              move.b  strlen,(straddr)        set new string length
000000B6 2885              move.l  newindex,(aindex)               set new index

000000B8 47F1 6000         lea     0(straddr,index),sindex

000000BC 9E42              sub     intlen,fieldwidth
000000BE 6708              beq.s   nospaces
000000C0 16FC 0020 space   move.b  #' ',(sindex)+
000000C4 5347              subq    #1,fieldwidth
000000C6 6EF8              bgt.s   space
          0000 00C8 nospaces equ *

000000C8 4A00              tst.b  sign
000000CA 6704              beq.s   nosgn
000000CC 16FC 002D         move.b  #'-',(sindex)+
          0000 00D0 nosgn   equ *

000000D0 16DA      digit   move.b  (dindex)+,(sindex)+
000000D2 5341              subq    #1,digits
000000D4 6EFA              bgt.s   digit

000000D6 42AD FFEA         clr.l   ioresult(a5)
000000DA 4E5E      ok      unlk    a6
000000DC 4ED0              jmp     (return)

000000DE 701C      error   moveq   #istrovfl,d0
000000E0 2B40 FFEA         move.l  d0,ioresult(a5)
000000E4 60F4              bra.s   ok

                           nosyms
                           end
```



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

Description

STROKES provides the stroke table for the DGL text labelling.

```
PAGE   1 [3.0] 12/28/84 22:27:16  ASSEMBLY OF STROKES.TEXT                                  *** File name: STROKES ***

PASS 1 COMPLETE. ERRORS: 0
    1                           *
    2                           * Graphics Low End
    3                           *
    4                           * Module       = STROKES
    5                           * Programer    = BJS
    6                           * Date         = 9/30/82
    7                           *
    8                           * Purpose : To provide stroke tables
    9                           *
   10                           *  Rev history
   11                           *
   12                           *    Created  - 9/30/82
   13                           *    Modified -
   14                           *
   15                           *
   16                           *     (c) Copyright Hewlett-Packard Company, 1983.
   17                           *     All rights are reserved.  Copying or other
   18                           *     reproduction of this program except for archival
   19                           *     purposes is prohibited without the prior
   20                           *     written consent of Hewlett-Packard Company.
   21                           *
   22                           *
   23                           *                  RESTRICTED RIGHTS LEGEND
   24                           *
   25                           *     Use, duplication, or disclosure by the Government
   26                           *     is subject to restrictions as set forth in
   27                           *     paragraph (b) (3) (B) of the Rights in Technical
   28                           *     Data and Computer Software clause in
   29                           *     DAR 7-104.9(a).
   30                           *
   31                           *     HEWLETT-PACKARD COMPANY
   32                           *     Fort Collins, Colorado
   33                           *
   34                           *
   35                                 mname GLE_STROKE
   36
   37    00000000                     rorg 0
   38                                 nosyms
   39
   40                                 def GLE_GLE_STROKE_TABLE
   41
   42              0000 0000 GLE_GLE_STROKE_TABLE equ *
   43
   44    00000000 0000 0018           DC.L  STD_CHAR-GLE_GLE_STROKE_TABLE
   45    00000004 0000 02B8           DC.L  STD_POINTER-GLE_GLE_STROKE_TABLE
   46
   47    00000008 0000 037A           DC.L  ROMAN_CHAR-GLE_GLE_STROKE_TABLE
   48    0000000C 0000 0556           DC.L  ROMAN_POINTER-GLE_GLE_STROKE_TABLE
   49
   50    00000010 0000 05C6           DC.L  KATA_CHAR-GLE_GLE_STROKE_TABLE
   51    00000014 0000 074E           DC.L  KATA_POINTER-GLE_GLE_STROKE_TABLE
   52
   53              0000 0018 std_char EQU *
   54
   55    00000018 44C4 46CC  DC.L $44C446CC
   56    0000001C 3ABC 5CDA  DC.L $3ABC5CDA
   57    00000020 34BC 5CD4  DC.L $34BC5CD4
   58    00000024 17F7 7999  DC.L $17F77999
```

```
PAGE   2 [3.0] 12/28/84 22:27:16  ASSEMBLY OF STROKES.TEXT                                  *** File name: STROKES ***

   59    00000028 25D5 E6E7  DC.L $25D5E6E7
   60    0000002C D8B8 A9AA  DC.L $D8B8A9AA
   61    00000030 BBEB 4CC4  DC.L $BBEB4CC4
   62    00000034 64F5 E6D5  DC.L $64F5E6D5
   63    00000038 E415 FB2A  DC.L $E415FB2A
   64    0000003C BBAC 9BAA  DC.L $BBAC9BAA
   65    00000040 749A 9BAC  DC.L $749A9BAC
   66    00000044 BCCB CA97  DC.L $BCCBCA97
   67    00000048 95A4 B4E7  DC.L $95A4B4E7
   68    0000004C 3ADC 54B6  DC.L $3ADC54B6
   69    00000050 BADC 34D6  DC.L $BADC34D6
   70    00000054 DABC 2BE5  DC.L $DABC2BE5
   71    00000058 6BA5 18F8  DC.L $6BA518F8
   72    0000005C 45CB 18F8  DC.L $45CB18F8
   73    00000060 44B4 B5C5  DC.L $44B4B5C5
   74    00000064 C3B2 18F8  DC.L $C3B218F8
   75    00000068 44C5 B5B4  DC.L $44C5B5B4
   76    0000006C C415 FB15  DC.L $C415FB15
   77    00000070 FBF5 E4A4  DC.L $FBF5E4A4
   78    00000074 959B ACEC  DC.L $959BACEC
   79    00000078 FB3B CCC4  DC.L $FB3BCCC4
   80    0000007C 34D4 1BAC  DC.L $34D41BAC
   81    00000080 DCEB E9D8  DC.L $DCEBE9D8
   82    00000084 B896 94E4  DC.L $B89694E4
   83    00000088 1BAC DCEB  DC.L $1BACDCEB
   84    0000008C E9D8 B858  DC.L $E9D8B858
   85    00000090 E7E5 D4A4  DC.L $E7E5D4A4
   86    00000094 9554 DC98  DC.L $9554DC98
   87    00000098 97E7 15A4  DC.L $97E715A4
   88    0000009C D4E5 E8D9  DC.L $D4E5E8D9
   89    000000A0 999C EC18  DC.L $999CEC18
   90    000000A4 D8E7 E5D4  DC.L $D8E7E5D4
   91    000000A8 A495 9BAC  DC.L $A4959BAC
   92    000000AC DCEB 1CEC  DC.L $DCEB1CEC
   93    000000B0 EAA6 A428  DC.L $EAA6A428
   94    000000B4 999B ACDC  DC.L $999BACDC
   95    000000B8 EBE9 D8A8  DC.L $EBE9D8A8
   96    000000BC 9795 A4D4  DC.L $9795A4D4
   97    000000C0 E5E7 D815  DC.L $E5E7D815
   98    000000C4 A4D4 E5EB  DC.L $A4D4E5EB
   99    000000C8 DCAC 9B99  DC.L $DCAC9B99
  100    000000CC A8E8 48B8  DC.L $A8E848B8
  101    000000D0 B9C9 C844  DC.L $B9C9C844
  102    000000D4 B4B5 C5C4  DC.L $B4B5C5C4
  103    000000D8 48B8 B9C9  DC.L $48B8B9C9
  104    000000DC C844 B4B5  DC.L $C844B4B5
  105    000000E0 C5C3 B255  DC.L $C5C3B255
  106    000000E4 A8DB 17F7  DC.L $A8DB17F7
  107    000000E8 19F9 25D8  DC.L $19F925D8
  108    000000EC AB1A 9BAC  DC.L $AB1A9BAC
  109    000000F0 DCEB EAB7  DC.L $DCEBEAB7
  110    000000F4 B634 B457  DC.L $B634B457
  111    000000F8 D9CA B9B8  DC.L $D9CAB9B8
  112    000000FC C7E7 F8FA  DC.L $C7E7F8FA
  113    00000100 DCBC 9A96  DC.L $DCBC9A96
  114    00000104 B4E4 F514  DC.L $B4E4F514
  115    00000108 9EAC ECFB  DC.L $9EACECFB
```

```
  116    0000010C F478 9824  DC.L $F4789824
  117    00000110 AC1C ECFB  DC.L $AC1CECFB
  118    00000114 F9E8 A814  DC.L $F9E8A814
  119    00000118 E4F5 F7E8  DC.L $E4F5F7E8
  120    0000011C 7BEC BC9A  DC.L $7BECBC9A
  121    00000120 96B4 E4F5  DC.L $96B4E4F5
  122    00000124 14E4 F5FB  DC.L $14E4F5FB
  123    00000128 EC9C 2CA4  DC.L $EC9C2CA4
  124    0000012C 18C8 7C9C  DC.L $18C87C9C
  125    00000130 94F4 149C  DC.L $94F4149C
  126    00000134 FC18 C87B  DC.L $FC18C87B
  127    00000138 ECBC 9A96  DC.L $ECBC9A96
  128    0000013C B4F4 F7D7  DC.L $B4F4F7D7
  129    00000140 149C 18F8  DC.L $149C18F8
  130    00000144 7CF4 2CEC  DC.L $7CF42CEC
  131    00000148 4CC4 24E4  DC.L $4CC424E4
  132    0000014C 15A4 C4D5  DC.L $15A4C4D5
  133    00000150 DC3C FC14  DC.L $DC3CFC14
  134    00000154 9C18 B87C  DC.L $9C18B87C
  135    00000158 B8F4 1C94  DC.L $B8F41C94
  136    0000015C F414 9CC8  DC.L $F4149CC8
  137    00000160 FCF4 149C  DC.L $FCF4149C
  138    00000164 F4FC 5CBC  DC.L $F4FC5CBC
  139    00000168 9A96 B4D4  DC.L $9A96B4D4
  140    0000016C F6FA DC14  DC.L $F6FADC14
  141    00000170 9CEC FBF9  DC.L $9CECFBF9
  142    00000174 E898 5CBC  DC.L $E8985CBC
  143    00000178 9A96 B4D4  DC.L $9A96B4D4
  144    0000017C F6FA DC56  DC.L $F6FADC56
  145    00000180 F414 9CEC  DC.L $F4149CEC
  146    00000184 FBF9 E898  DC.L $FBF9E898
  147    00000188 38F4 15A4  DC.L $38F415A4
  148    0000018C E4F5 F7E8  DC.L $E4F5F7E8
  149    00000190 A899 9BAC  DC.L $A8999BAC
  150    00000194 ECFB 1CFC  DC.L $ECFB1CFC
  151    00000198 4CC4 1C95  DC.L $4CC41C95
  152    0000019C A4E4 F5FC  DC.L $A4E4F5FC
  153    000001A0 1C9B C4FB  DC.L $1C9BC4FB
  154    000001A4 FC1C 94C9  DC.L $FC1C94C9
  155    000001A8 F4FC 14FC  DC.L $F4FC14FC
  156    000001AC 1CF4 1CC8  DC.L $1CF41CC8
  157    000001B0 FC48 C41C  DC.L $FC48C41C
  158    000001B4 FC94 F46C  DC.L $FC94F46C
  159    000001B8 BCB4 E41B  DC.L $BCB4E41B
  160    000001BC F52C DCD4  DC.L $F52CDCD4
  161    000001C0 A417 CAF7  DC.L $A417CAF7
  162    000001C4 00F0 3CDA  DC.L $00F03CDA
  163    000001C8 29D9 E8E4  DC.L $29D9E8E4
  164    000001CC 67B7 A6A5  DC.L $67B7A6A5
  165    000001D0 B4F4 1C94  DC.L $B4F41C94
  166    000001D4 17B9 D9E8  DC.L $17B9D9E8
  167    000001D8 E5D4 B496  DC.L $E5D4B496
  168    000001DC 68D9 A998  DC.L $68D9A998
  169    000001E0 95A4 D4E5  DC.L $95A4D4E5
  170    000001E4 6CE4 67C9  DC.L $6CE467C9
  171    000001E8 A998 95A4  DC.L $A99895A4
  172    000001EC C4E6 17E7  DC.L $C4E617E7
```



```
  173    000001F0 E8D9 A998  DC.L $E8D9A998
  174    000001F4 95A4 D4E5  DC.L $95A4D4E5
  175    000001F8 6BDC CCBB  DC.L $6BDCCCBB
  176    000001FC B418 D867  DC.L $B418D867
  177    00000200 C9A9 9895  DC.L $C9A99895
  178    00000204 A4C4 E669  DC.L $A4C4E669
  179    00000208 E2D1 A192  DC.L $E2D1A192
  180    0000020C 1C94 17B9  DC.L $1C9417B9
  181    00000210 D9E8 E439  DC.L $D9E8E439
  182    00000214 C9C4 4BCB  DC.L $C9C44BCB
  183    00000218 59E9 E2D1  DC.L $59E9E2D1
  184    0000021C A192 6BEB  DC.L $A1926BEB
  185    00000220 1C94 5995  DC.L $1C945995
  186    00000224 37E4 3CCC  DC.L $37E43CCC
  187    00000228 C419 9418  DC.L $C4199418
  188    0000022C A9B9 C8C4  DC.L $A9B9C8C4
  189    00000230 48D9 E9F8  DC.L $48D9E9F8
  190    00000234 F414 9917  DC.L $F4149917
  191    00000238 B9D9 E8E4  DC.L $B9D9E8E4
  192    0000023C 1895 A4D4  DC.L $1895A4D4
  193    00000240 E5E8 D9A9  DC.L $E5E8D9A9
  194    00000244 9811 9917  DC.L $98119917
  195    00000248 B9D9 E8E5  DC.L $B9D9E8E5
  196    0000024C D4B4 9667  DC.L $D4B49667
  197    00000250 D9A9 9895  DC.L $D9A99895
  198    00000254 A4C4 E669  DC.L $A4C4E669
  199    00000258 E129 A427  DC.L $E129A427
  200    0000025C C9D9 E868  DC.L $C9D9E868
  201    00000260 D9A9 98E5  DC.L $D9A998E5
  202    00000264 D4A4 953B  DC.L $D4A4953B
  203    00000268 B5C4 D4E5  DC.L $B5C4D4E5
  204    0000026C 19D9 1995  DC.L $19D91995
  205    00000270 A4C4 E669  DC.L $A4C4E669
  206    00000274 E419 C4F9  DC.L $E419C4F9
  207    00000278 1995 A4B4  DC.L $1995A4B4
  208    0000027C C5C8 45D4  DC.L $C5C845D4
  209    00000280 E4F5 F919  DC.L $E4F5F919
  210    00000284 E469 9419  DC.L $E4699419
  211    00000288 C479 A191  DC.L $C479A191
  212    0000028C 19E9 94E4  DC.L $19E994E4
  213    00000290 6CDC CBC9  DC.L $6CDCCBC9
  214    00000294 B8C7 C5D4  DC.L $B8C7C5D4
  215    00000298 E44F C02C  DC.L $E44FC02C
  216    0000029C BCCB C9D8  DC.L $BCCBC9D8
  217    000002A0 C7C5 B4A4  DC.L $C7C5B4A4
  218    000002A4 18A9 B9D7  DC.L $18A9B9D7
  219    000002A8 E7F8 1BAC  DC.L $E7F81BAC
  220    000002AC 4C99 17EC  DC.L $4C9917EC
  221    000002B0 7B95 24F9  DC.L $7B9524F9
  222    000002B4 77C4 64F5  DC.L $77C464F5
  223
  224              0000 02B8 std_pointer EQU *
  225
  226    000002B8 0001       DC.W 1
  227    000002BA 0001       DC.W 1
  228    000002BC 0005       DC.W 5
  229    000002BE 0009       DC.W 9
```

```
000002C0 0011       DC.W 17
000002C2 001D       DC.W 29
000002C4 0029       DC.W 41
000002C6 0035       DC.W 53
000002C8 0037       DC.W 55
000002CA 003B       DC.W 59
000002CC 003F       DC.W 63
000002CE 0045       DC.W 69
000002D0 0049       DC.W 73
000002D2 004F       DC.W 79
000002D4 0051       DC.W 81
000002D6 0056       DC.W 86
000002D8 0058       DC.W 88
000002DA 0062       DC.W 98
000002DC 0067       DC.W 103
000002DE 0071       DC.W 113
000002E0 007E       DC.W 126
000002E2 0083       DC.W 131
000002E4 008C       DC.W 140
000002E6 0097       DC.W 151
000002E8 009C       DC.W 156
000002EA 00AC       DC.W 172
000002EC 00B7       DC.W 183
000002EE 00C1       DC.W 193
000002F0 00CC       DC.W 204
000002F2 00CF       DC.W 207
000002F4 00D3       DC.W 211
000002F6 00D6       DC.W 214
000002F8 00E0       DC.W 224
000002FA 00F0       DC.W 240
000002FC 00F8       DC.W 248
000002FE 0105       DC.W 261
00000300 010D       DC.W 269
00000302 0115       DC.W 277
00000304 011B       DC.W 283
00000306 0120       DC.W 288
00000308 0129       DC.W 297
0000030A 012F       DC.W 303
0000030C 0135       DC.W 309
0000030E 013C       DC.W 316
00000310 0143       DC.W 323
00000312 0146       DC.W 326
00000314 014B       DC.W 331
00000316 014F       DC.W 335
00000318 0158       DC.W 344
0000031A 015F       DC.W 351
0000031C 016A       DC.W 362
0000031E 0173       DC.W 371
00000320 017F       DC.W 383
00000322 0183       DC.W 387
00000324 0189       DC.W 393
00000326 018E       DC.W 398
00000328 0193       DC.W 403
0000032A 0197       DC.W 407
0000032C 019C       DC.W 412
0000032E 01A0       DC.W 416
00000330 01A4       DC.W 420
```



```
00000332 01A6       DC.W 422
00000334 01AA       DC.W 426
00000336 01AD       DC.W 429
00000338 01AF       DC.W 431
0000033A 01B1       DC.W 433
0000033C 01BB       DC.W 443
0000033E 01C5       DC.W 453
00000340 01CD       DC.W 461
00000342 01D7       DC.W 471
00000344 01E1       DC.W 481
00000346 01E8       DC.W 488
00000348 01F5       DC.W 501
0000034A 01FC       DC.W 508
0000034C 0201       DC.W 513
0000034E 0209       DC.W 521
00000350 020F       DC.W 527
00000352 0212       DC.W 530
00000354 021E       DC.W 542
00000356 0225       DC.W 549
00000358 022E       DC.W 558
0000035A 0238       DC.W 568
0000035C 0242       DC.W 578
0000035E 0248       DC.W 584
00000360 0250       DC.W 592
00000362 0257       DC.W 599
00000364 025E       DC.W 606
00000366 0261       DC.W 609
00000368 026C       DC.W 620
0000036A 0270       DC.W 624
0000036C 0275       DC.W 629
0000036E 0279       DC.W 633
00000370 0282       DC.W 642
00000372 0284       DC.W 644
00000374 028D       DC.W 653
00000376 0293       DC.W 659
00000378 02A1       DC.W 673

            0000 037A roman_char equ *

0000037A 3BDD 2DCB  DC.L $3BDD2DCB
0000037E 2BCD EB2B  DC.L $2BCDEB2B
00000382 AB5B DB1D  DC.L $AB5BDB1D
00000386 AEBE DCEC  DC.L $AEBEDCEC
0000038A FD6A EBDC  DC.L $FD6AEBDC
0000038E BCAB A414  DC.L $BCABA414
00000392 F416 B618  DC.L $F416B618
00000396 B81E FE3C  DC.L $B81EFE3C
0000039A AC9B 9AA9  DC.L $AC9B9AA9
0000039E B9CA CBBC  DC.L $B9CACBBC
000003A2 68D9 A998  DC.L $68D9A998
000003A6 95A4 D4E5  DC.L $95A4D4E5
000003AA 44C3 D2C1  DC.L $44C3D2C1
000003AE B114 9BF4  DC.L $B1149BF4
000003B2 FB1D AEBE  DC.L $FB1DAEBE
000003B6 DCEC FD14  DC.L $DCECFD14
000003BA 9917 B9D9  DC.L $9917B9D9
000003BE E9E4 1CAD  DC.L $E8E41CAD
```

```
000003C2 BDCC DCED  DC.L $BDCCDCED
000003C6 44CA 4CCC  DC.L $44CA4CCC
000003CA 76F5 E4B4  DC.L $76F5E4B4
000003CE A5A6 D9DA  DC.L $A5A6D9DA
000003D2 5CDC 5ABA  DC.L $5CDC5ABA
000003D6 A9A7 B6D6  DC.L $A9A7B6D6
000003DA E7E9 DAFC  DC.L $E7E9DAFC
000003DE 3A9C 3694  DC.L $3A9C3694
000003E2 56F4 6AEB  DC.L $56F46AEB
000003E6 DCBC ABA4  DC.L $DCBCABA4
000003EA 14F4 17B7  DC.L $14F417B7
000003EE 6BDC BCAB  DC.L $6BDCBCAB
000003F2 D8C7 49B8  DC.L $D8C749B8
000003F6 E5D4 B4A5  DC.L $E5D4B4A5
000003FA 29D9 E8E4  DC.L $29D9E8E4
000003FE 67B7 A6A5  DC.L $67B7A6A5
00000402 B4F4 2BCD  DC.L $B4F42BCD
00000406 EB17 E7E8  DC.L $EB17E7E8
0000040A D9A9 9895  DC.L $D9A99895
0000040E A4D4 E52B  DC.L $A4D4E52B
00000412 CDEB 59A9  DC.L $CDEB59A9
00000416 9895 A4D4  DC.L $9895A4D4
0000041A E5E8 D92B  DC.L $E5E8D92B
0000041E CDEB 1995  DC.L $CDEB1995
00000422 A4C4 E669  DC.L $A4C4E669
00000426 E42B CDEB  DC.L $E42BCDEB
0000042A 29D9 E8E4  DC.L $29D9E8E4
0000042E 67B7 A6A5  DC.L $67B7A6A5
00000432 B4F4 3BDD  DC.L $B4F43BDD
00000436 17E7 E8D9  DC.L $17E7E8D9
0000043A A998 95A4  DC.L $A99895A4
0000043E D4E5 3BDD  DC.L $D4E53BDD
00000442 59A9 9895  DC.L $59A99895
00000446 A4D4 E5E8  DC.L $A4D4E5E8
0000044A D93B DD19  DC.L $D93BDD19
0000044E 95A4 C4E6  DC.L $95A4C4E6
00000452 69E4 3BDD  DC.L $69E43BDD
00000456 29D9 E8E4  DC.L $29D9E8E4
0000045A 67B7 A6A5  DC.L $67B7A6A5
0000045E B4F4 3DDB  DC.L $B4F43DDB
00000462 17E7 E8D9  DC.L $17E7E8D9
00000466 A998 95A4  DC.L $A99895A4
0000046A D4E5 2DCB  DC.L $D4E52DCB
0000046E 59A9 9895  DC.L $59A99895
00000472 A4D4 E5E8  DC.L $A4D4E5E8
00000476 D92D CB19  DC.L $D92DCB19
0000047A 95A4 C4E6  DC.L $95A4C4E6
0000047E 69E4 2DCB  DC.L $69E42DCB
00000482 29D9 E8E4  DC.L $29D9E8E4
00000486 67B7 A6A5  DC.L $67B7A6A5
0000048A B4F4 2BAB  DC.L $B4F42BAB
0000048E 5BDB 17E7  DC.L $5BDB17E7
00000492 E8D9 A998  DC.L $E8D9A998
00000496 95A4 D4E5  DC.L $95A4D4E5
0000049A 2BAB 5BDB  DC.L $2BAB5BDB
0000049E 59A9 9895  DC.L $59A99895
000004A2 A4D4 E5E8  DC.L $A4D4E5E8
```



```
000004A6 D92B AB5B  DC.L $D92BAB5B
000004AF DB19 95A4  DC.L $DB1995A4
000004AE C4E6 69E4  DC.L $C4E669E4
000004B2 2BAB 5BDB  DC.L $2BAB5BDB
000004B6 149B ACEC  DC.L $149BACEC
000004BA FBF4 7898  DC.L $FBF47898
000004BE 4DBE CFDE  DC.L $4DBECFDE
000004C2 CD39 C9C4  DC.L $CD39C9C4
000004C6 2BCD EB3C  DC.L $2BCDEB3C
000004CA DCFA F6D4  DC.L $DCFAF6D4
000004CE B496 9ABC  DC.L $B4969ABC
000004D2 7C94 149B  DC.L $7C94149B
000004D6 ACFC 4CC4  DC.L $ACFC4CC4
000004DA F417 C748  DC.L $F417C748
000004DE E829 D9E8  DC.L $E829D9E8
000004E2 E467 87A6  DC.L $E467B7A6
000004E6 A5B4 F44A  DC.L $A5B4F44A
000004EA BBCC DBCA  DC.L $BBCCDBCA
000004EE 39C9 C44B  DC.L $39C9C44B
000004F2 ED59 A998  DC.L $ED59A998
000004F6 95A4 D4E5  DC.L $95A4D4E5
000004FA E8D9 6994  DC.L $E8D96994
000004FE 19B9 C8C5  DC.L $19B9C8C5
00000502 B4A4 9598  DC.L $B4A49598
00000506 A7F7 F8E9  DC.L $A7F7F8E9
0000050A D9C8 45D4  DC.L $D9C845D4
0000050E E4F5 149B  DC.L $E4F5149B
00000512 ACEC FBF4  DC.L $ACECFBF4
00000516 7898 2EAE  DC.L $78982EAE
0000051A 6EEE 39C9  DC.L $6EEE39C9
0000051E C42D CB3C  DC.L $C42DCB3C
00000522 DCFA F6D4  DC.L $DCFAF6D4
00000526 B496 9ABC  DC.L $B4969ABC
0000052A 2EAE 6EEE  DC.L $2EAE6EEE
0000052E 1C95 A4E4  DC.L $1C95A4E4
00000532 F5FC 2EAE  DC.L $F5FC2EAE
00000536 6EEE 7C9C  DC.L $6EEE7C9C
0000053A 94F4 18C8  DC.L $94F418C8
0000053E 4DDE 39C9  DC.L $4DDE39C9
00000542 C42B AB5B  DC.L $C42BAB5B
00000546 DB14 9BAC  DC.L $DB149BAC
0000054A CCDB DAB8  DC.L $CCDBDAB8
0000054E D8E7 E5D4  DC.L $D8E7E5D4
00000552 B400 0000  DC.L $B4000000

          0000 0556 roman_pointer equ *

00000556 0001       DC.W 1
00000558 0003       DC.W 3
0000055A 0005       DC.W 5
0000055C 0008       DC.W 8
0000055E 000C       DC.W 12
00000560 0012       DC.W 18
00000562 0012       DC.W 18
00000564 0012       DC.W 18
00000566 001E       DC.W 30
00000568 0020       DC.W 32
```

```
0000056A 0020        DC.W 32
0000056C 0020        DC.W 32
0000056E 0029        DC.W 41
00000570 0029        DC.W 41
00000572 0036        DC.W 54
00000574 0040        DC.W 64
00000576 004D        DC.W 77
00000578 0051        DC.W 81
0000057A 005B        DC.W 91
0000057C 006B        DC.W 107
0000057E 0075        DC.W 117
00000580 0075        DC.W 117
00000582 0081        DC.W 129
00000584 0081        DC.W 129
00000586 0081        DC.W 129
00000588 008E        DC.W 142
0000058A 009B        DC.W 155
0000058C 00A7        DC.W 167
0000058E 00B1        DC.W 177
00000590 00BD        DC.W 189
00000592 00C9        DC.W 201
00000594 00D4        DC.W 212
00000596 00DD        DC.W 221
00000598 00E9        DC.W 233
0000059A 00F5        DC.W 245
0000059C 0100        DC.W 256
0000059E 0109        DC.W 265
000005A0 0117        DC.W 279
000005A2 0125        DC.W 293
000005A4 0132        DC.W 306
000005A6 013D        DC.W 317
000005A8 014A        DC.W 330
000005AA 0150        DC.W 336
000005AC 015B        DC.W 347
000005AE 0166        DC.W 358
000005B0 0175        DC.W 373
000005B2 017A        DC.W 378
000005B4 0185        DC.W 389
000005B6 0197        DC.W 407
000005B8 01A3        DC.W 419
000005BA 01A8        DC.W 424
000005BC 01B5        DC.W 437
000005BE 01BF        DC.W 447
000005C0 01C7        DC.W 455
000005C2 01CE        DC.W 462
000005C4 01DA        DC.W 474

             0000 05C6 kata_char equ *

000005C6 2796 95A4   DC.L $279695A4
000005CA B4C5 C6B7   DC.L $B4C5C6B7
000005CE A76C BCB8   DC.L $A76CBCB8
000005D2 24D4 D826   DC.L $24D4D826
000005D6 C447 C8B8   DC.L $C447C8B8
000005DA B7C7 1BFB   DC.L $B7C71BFB
000005DE F7C4 B418   DC.L $F7C4B418
000005E2 F829 E9E8   DC.L $F829E9E8
```



```
000005E6 C647 C5B4   DC.L $C647C5B4
000005EA 59A6 48C4   DC.L $59A648C4
000005EE 27A8 E8E6   DC.L $27A8E8E6
000005F2 C448 C938   DC.L $C448C938
000005F6 D848 C424   DC.L $D848C424
000005FA E428 E859   DC.L $E428E859
000005FE D457 A428   DC.L $D457A428
00000602 E8E7 D639   DC.L $E8E7D639
00000606 B438 D8D4   DC.L $B438D8D4
0000060A 24F4 28E8   DC.L $24F428E8
0000060E E4A4 36E6   DC.L $E4A436E6
00000612 28A7 48C7   DC.L $28A748C7
00000616 68E6 C418   DC.L $68E6C418
0000061A F81B FBF9   DC.L $F81BFBF9
0000061E C749 C5B4   DC.L $C749C5B4
00000622 6C97 4AC4   DC.L $6C974AC4
00000626 189A FAF7   DC.L $189AFAF7
0000062A C44C 4CCA   DC.L $C44C4CCA
0000062E 2BEB 4BC5   DC.L $2BEB4BC5
00000632 15F5 5CD4   DC.L $15F55CD4
00000636 1AFA 5995   DC.L $1AFA5995
0000063A 1AFA F5E4   DC.L $1AFAF5E4
0000063E 3CB6 944C   DC.L $3CB6944C
00000642 C46A AA17   DC.L $C46AAA17
00000646 F719 BBFB   DC.L $F719BBFB
0000064A F8B4 2CAA   DC.L $F8B42CAA
0000064E 992A FA5A   DC.L $992AFA5A
00000652 D5B4 1BFB   DC.L $D5B41BFB
00000656 F595 2CA8   DC.L $F5952CA8
0000065A 1AFA 6CE7   DC.L $1AFA6CE7
0000065E B41B BB18   DC.L $B41BBB18
00000662 B814 C4F7   DC.L $B814C4F7
00000666 F91B FBFA   DC.L $F91BFBFA
0000066A 9447 F41A   DC.L $9447F41A
0000066E FAF9 D72C   DC.L $FAF9D72C
00000672 A5B4 F474   DC.L $A5B4F474
00000676 1BB9 7BF8   DC.L $1BB97BF8
0000067A B418 BABB   DC.L $B418BABB
0000067E FBF8 B438   DC.L $FBF8B438
00000682 D62B DBEC   DC.L $D62BDBEC
00000686 4BC6 A418   DC.L $4BC6A418
0000068A F81B 9949   DC.L $F81B9949
0000068E CB7B F7C4   DC.L $CB7BF7C4
00000692 2BEB 7999   DC.L $2BEB7999
00000696 49C6 A43C   DC.L $49C6A43C
0000069A B43A E74C   DC.L $B43AE74C
0000069E C6A4 1AFA   DC.L $C6A41AFA
000006A2 2BEB 15F5   DC.L $2BEB15F5
000006A6 1BFB FAFA   DC.L $1BFBFAFA
000006AA 9438 E51B   DC.L $9438E51B
000006AE FB95 44C8   DC.L $FB9544C8
000006B2 F54C CB6C   DC.L $F54CCB6C
000006B6 E8A4 14A5   DC.L $E8A414A5
000006BA AA5B F9F4   DC.L $AA5BF9F4
000006BE 1C95 A4F4   DC.L $1C95A4F4
000006C2 19E9 1BFB   DC.L $19E91BFB
000006C6 F8B4 A419   DC.L $F8B4A419
```

```
  572     000006CA BBF7 F64C   DC.L $BBF7F64C
  573     000006CE C41A FA27   DC.L $C41AFA27
  574     000006D2 A695 67F6   DC.L $A69567F6
  575     000006D6 F51B FBF9   DC.L $F51BFBF9
  576     000006DA C637 E42C   DC.L $C637E42C
  577     000006DE BBDB EA29   DC.L $BBDBEA29
  578     000006E2 B8D8 E726   DC.L $B8D8E726
  579     000006E6 B5D5 E45B   DC.L $B5D5E45B
  580     000006EA 9795 F576   DC.L $9795F576
  581     000006EE 76F4 7BFA   DC.L $76F47BFA
  582     000006F2 9429 E51B   DC.L $9429E51B
  583     000006F6 FB18 F83B   DC.L $FB18F83B
  584     000006FA B5C4 F43C   DC.L $B5C4F43C
  585     000006FE B41A FAF9   DC.L $B41AFAF9
  586     00000702 F9D7 2BDB   DC.L $F9D72BDB
  587     00000706 D515 F51B   DC.L $D515F51B
  588     0000070A FBF5 9528   DC.L $FBF59528
  589     0000070E F82B EB19   DC.L $F82BEB19
  590     00000712 F9F7 C4C4   DC.L $F9F7C4C4
  591     00000716 2BA7 6BE6   DC.L $2BA76BE6
  592     0000071A C42B A594   DC.L $C42BA594
  593     0000071E 4BC4 D4F6   DC.L $4BC4D4F6
  594     00000722 F72B A4C4   DC.L $F72BA4C4
  595     00000726 F775 959B   DC.L $F775959B
  596     0000072A FBF5 199B   DC.L $FBF5199B
  597     0000072E FBF7 C4B4   DC.L $FBF7C4B4
  598     00000732 1ABA 7AF9   DC.L $1ABA7AF9
  599     00000736 B494 399B   DC.L $B494399B
  600     0000073A 3CDA 2C9B   DC.L $3CDA2C9B
  601     0000073E 9AA9 B9CA   DC.L $9AA9B9CA
  602     00000742 CBBC AC1C   DC.L $CBBCAC1C
  603     00000746 C9FC 49C4   DC.L $C9FC49C4
  604     0000074A 68A8 26E6   DC.L $68A826E6
  605
  606               0000 074E kata_pointer EQU *
  607
  608     0000074E 0001        DC.W 1
  609     00000750 000A        DC.W 10
  610     00000752 000D        DC.W 13
  611     00000754 0010        DC.W 16
  612     00000756 0012        DC.W 18
  613     00000758 0017        DC.W 23
  614     0000075A 001E        DC.W 30
  615     0000075C 0025        DC.W 37
  616     0000075E 0029        DC.W 41
  617     00000760 0030        DC.W 48
  618     00000762 0036        DC.W 54
  619     00000764 003C        DC.W 60
  620     00000766 0042        DC.W 66
  621     00000768 0047        DC.W 71
  622     0000076A 004D        DC.W 77
  623     0000076C 0054        DC.W 84
  624     0000076E 0056        DC.W 86
  625     00000770 005D        DC.W 93
  626     00000772 0061        DC.W 97
  627     00000774 0069        DC.W 105
  628     00000776 006F        DC.W 111
```



```
  629     00000778 0075        DC.W 117
  630     0000077A 007C        DC.W 124
  631     0000077C 0082        DC.W 130
  632     0000077E 0087        DC.W 135
  633     00000780 008F        DC.W 143
  634     00000782 0093        DC.W 147
  635     00000784 009A        DC.W 154
  636     00000786 00A2        DC.W 162
  637     00000788 00A8        DC.W 168
  638     0000078A 00B1        DC.W 177
  639     0000078C 00B6        DC.W 182
  640     0000078E 00BE        DC.W 190
  641     00000790 00C6        DC.W 198
  642     00000792 00CD        DC.W 205
  643     00000794 00D4        DC.W 212
  644     00000796 00D8        DC.W 216
  645     00000798 00DD        DC.W 221
  646     0000079A 00E1        DC.W 225
  647     0000079C 00E8        DC.W 232
  648     0000079E 00F0        DC.W 240
  649     000007A0 00F3        DC.W 243
  650     000007A2 00F9        DC.W 249
  651     000007A4 00FF        DC.W 255
  652     000007A6 0104        DC.W 260
  653     000007A8 0108        DC.W 264
  654     000007AA 0112        DC.W 274
  655     000007AC 0118        DC.W 280
  656     000007AE 0124        DC.W 292
  657     000007B0 012B        DC.W 299
  658     000007B2 0130        DC.W 304
  659     000007B4 0138        DC.W 312
  660     000007B6 013F        DC.W 319
  661     000007B8 0144        DC.W 324
  662     000007BA 014A        DC.W 330
  663     000007BC 0151        DC.W 337
  664     000007BE 0156        DC.W 342
  665     000007C0 015E        DC.W 350
  666     000007C2 0162        DC.W 354
  667     000007C4 0167        DC.W 359
  668     000007C6 016D        DC.W 365
  669     000007C8 0173        DC.W 371
  670     000007CA 0177        DC.W 375
  671     000007CC 0180        DC.W 384
  672     000007CE 0189        DC.W 393
  673
  674                          END
PASS 1 ERRORS: 0
PASS 2 ERRORS: 0
```

SYSDEF

Description

SYSDEF provides the symbol table for SYSTEM_P.

Usage

SYSDEF is accessed by the loader.

Notes

(This file was generated programmatically!)

```
PAGE   1 [3.0] 12/26/84 21:03:21  ASSEMBLY OF SYSDEF.TEXT                                   *** File name: SYSDEF ***

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 0724              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 [3.0] 12/26/84 21:03:21  ASSEMBLY OF SYSDEF.TEXT                                   *** File name: SYSDEF ***

   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_CACHE_OFF
   33    00000058 0D                 DC.B  13,'ASM_CACHE_OFF'
   33    00000059 41
   33    0000005A 53
   33    0000005B 4D
   33    0000005C 5F
   33    0000005D 43
   33    0000005E 41
   33    0000005F 43
   33    00000060 48
   33    00000061 45
   33    00000062 5F
   33    00000063 4F
   33    00000064 46
   33    00000065 46
   34    00000066 1206               DC.W  GVR
   35    00000068 0000 0000          DC.L  ASM_CACHE_OFF
   36                                REFR  ASM_CACHE_ON
   37    0000006C 0C                 DC.B  12,'ASM_CACHE_ON'
   37    0000006D 41
   37    0000006E 53
   37    0000006F 4D
   37    00000070 5F
   37    00000071 43
   37    00000072 41
   37    00000073 43
   37    00000074 48
   37    00000075 45
   37    00000076 5F
   37    00000077 4F
   37    00000078 4E
   38    0000007A 1206               DC.W  GVR
   39    0000007C 0000 0000          DC.L  ASM_CACHE_ON
   40                                REFR  ASM_CI_SWITCH
   41    00000080 0D                 DC.B  13,'ASM_CI_SWITCH'
   41    00000081 41
   41    00000082 53
   41    00000083 4D
   41    00000084 5F
   41    00000085 43
   41    00000086 49
```

```
41    00000087 5F
41    00000088 53
41    00000089 57
41    0000008A 49
41    0000008B 54
41    0000008C 43
41    0000008D 48
42    0000008E 1206          DC.W  GVR
43    00000090 0000 0000     DC.L  ASM_CI_SWITCH
44                           REFR  ASM_CPYMSG
45    00000094 0A            DC.B  10,'ASM_CPYMSG'
45    00000095 41
45    00000096 53
45    00000097 4D
45    00000098 5F
45    00000099 43
45    0000009A 50
45    0000009B 59
45    0000009C 4D
45    0000009D 53
45    0000009E 47
46    000000A0 1206          DC.W  GVR
47    000000A2 0000 0000     DC.L  ASM_CPYMSG
48                           REFR  ASM_DIFFERENCE
49    000000A6 0E            DC.B  14,'ASM_DIFFERENCE'
49    000000A7 41
49    000000A8 53
49    000000A9 4D
49    000000AA 5F
49    000000AB 44
49    000000AC 49
49    000000AD 46
49    000000AE 46
49    000000AF 45
49    000000B0 52
49    000000B1 45
49    000000B2 4E
49    000000B3 43
49    000000B4 45
50    000000B6 1206          DC.W  GVR
51    000000B8 0000 0000     DC.L  ASM_DIFFERENCE
52                           REFR  ASM_DIV
53    000000BC 07            DC.B  7,'ASM_DIV'
53    000000BD 41
53    000000BE 53
53    000000BF 4D
53    000000C0 5F
53    000000C1 44
53    000000C2 49
53    000000C3 56
54    000000C4 1206          DC.W  GVR
55    000000C6 0000 0000     DC.L  ASM_DIV
56                           REFR  ASM_EQUAL
57    000000CA 09            DC.B  9,'ASM_EQUAL'
57    000000CB 41
57    000000CC 53
57    000000CD 4D
```



```
57    000000CE 5F
57    000000CF 45
57    000000D0 51
57    000000D1 55
57    000000D2 41
57    000000D3 4C
58    000000D4 1206          DC.W  GVR
59    000000D6 0000 0000     DC.L  ASM_EQUAL
60                           REFR  ASM_ERRMSG
61    000000DA 0A            DC.B  10,'ASM_ERRMSG'
61    000000DB 41
61    000000DC 53
61    000000DD 4D
61    000000DE 5F
61    000000DF 45
61    000000E0 52
61    000000E1 52
61    000000E2 4D
61    000000E3 53
61    000000E4 47
62    000000E6 1206          DC.W  GVR
63    000000E8 0000 0000     DC.L  ASM_ERRMSG
64                           REFR  ASM_FASTMOVE
65    000000EC 0C            DC.B  12,'ASM_FASTMOVE'
65    000000ED 41
65    000000EE 53
65    000000EF 4D
65    000000F0 5F
65    000000F1 46
65    000000F2 41
65    000000F3 53
65    000000F4 54
65    000000F5 4D
65    000000F6 4F
65    000000F7 56
65    000000F8 45
66    000000FA 1206          DC.W  GVR
67    000000FC 0000 0000     DC.L  ASM_FASTMOVE
68                           REFR  ASM_FINDROMS
69    00000100 0C            DC.B  12,'ASM_FINDROMS'
69    00000101 41
69    00000102 53
69    00000103 4D
69    00000104 5F
69    00000105 46
69    00000106 49
69    00000107 4E
69    00000108 44
69    00000109 52
69    0000010A 4F
69    0000010B 4D
69    0000010C 53
70    0000010E 1206          DC.W  GVR
71    00000110 0000 0000     DC.L  ASM_FINDROMS
72                           REFR  ASM_FLPYINIT
73    00000114 0C            DC.B  12,'ASM_FLPYINIT'
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 4F
   73    0000011F 49
   73    00000120 54
   74    00000122 1206          DC.W  GVR
   75    00000124 0000 0000     DC.L  ASM_FLPYINIT
   76                           REFR  ASM_FLPYMREAD
   77    00000128 0D            DC.B  13,'ASM_FLPYMREAD'
   77    00000129 41
   77    0000012A 53
   77    0000012B 4D
   77    0000012C 5F
   77    0000012D 46
   77    0000012E 4C
   77    0000012F 50
   77    00000130 59
   77    00000131 4D
   77    00000132 52
   77    00000133 45
   77    00000134 41
   77    00000135 44
   78    00000136 1206          DC.W  GVR
   79    00000138 0000 0000     DC.L  ASM_FLPYMREAD
   80                           REFR  ASM_FLPYMWRITE
   81    0000013C 0E            DC.B  14,'ASM_FLPYMWRITE'
   81    0000013D 41
   81    0000013E 53
   81    0000013F 4D
   81    00000140 5F
   81    00000141 46
   81    00000142 4C
   81    00000143 50
   81    00000144 59
   81    00000145 4D
   81    00000146 57
   81    00000147 52
   81    00000148 49
   81    00000149 54
   81    0000014A 45
   82    0000014C 1206          DC.W  GVR
   83    0000014E 0000 0000     DC.L  ASM_FLPYMWRITE
   84                           REFR  ASM_FLPYREAD
   85    00000152 0C            DC.B  12,'ASM_FLPYREAD'
   85    00000153 41
   85    00000154 53
   85    00000155 4D
   85    00000156 5F
   85    00000157 46
   85    00000158 4C
   85    00000159 50
```



```
   85    0000015A 59
   85    0000015B 52
   85    0000015C 45
   85    0000015D 41
   85    0000015E 44
   86    00000160 1206          DC.W  GVR
   87    00000162 0000 0000     DC.L  ASM_FLPYREAD
   88                           REFR  ASM_FLPY_WRT
   89    00000166 0C            DC.B  12,'ASM_FLPY_WRT'
   89    00000167 41
   89    00000168 53
   89    00000169 4D
   89    0000016A 5F
   89    0000016B 46
   89    0000016C 4C
   89    0000016D 50
   89    0000016E 59
   89    0000016F 5F
   89    00000170 57
   89    00000171 52
   89    00000172 54
   90    00000174 1206          DC.W  GVR
   91    00000176 0000 0000     DC.L  ASM_FLPY_WRT
   92                           REFR  ASM_F_PWR_ON
   93    0000017A 0C            DC.B  12,'ASM_F_PWR_ON'
   93    0000017B 41
   93    0000017C 53
   93    0000017D 4D
   93    0000017E 5F
   93    0000017F 46
   93    00000180 5F
   93    00000181 50
   93    00000182 57
   93    00000183 52
   93    00000184 5F
   93    00000185 4F
   93    00000186 4E
   94    00000188 1206          DC.W  GVR
   95    0000018A 0000 0000     DC.L  ASM_F_PWR_ON
   96                           REFR  ASM_IN
   97    0000018E 06            DC.B  6,'ASM_IN'
   97    0000018F 41
   97    00000190 53
   97    00000191 4D
   97    00000192 5F
   97    00000193 49
   97    00000194 4E
   98    00000196 1206          DC.W  GVR
   99    00000198 0000 0000     DC.L  ASM_IN
  100                           REFR  ASM_INCLUSION
  101    0000019C 0D            DC.B  13,'ASM_INCLUSION'
  101    0000019D 41
  101    0000019E 53
  101    0000019F 4D
  101    000001A0 5F
  101    000001A1 49
  101    000001A2 4E
```

```
101    000001A3 43
101    000001A4 4C
101    000001A5 55
101    000001A6 53
101    000001A7 49
101    000001A8 4F
101    000001A9 4E
102    000001AA 1206          DC.W  GVR
103    000001AC 0000 0000     DC.L  ASM_INCLUSION
104                           REFR  ASM_INTERSECT
105    000001B0 0D            DC.B  13,'ASM_INTERSECT'
105    000001B1 41
105    000001B2 53
105    000001B3 4D
105    000001B4 5F
105    000001B5 49
105    000001B6 4E
105    000001B7 54
105    000001B8 45
105    000001B9 52
105    000001BA 53
105    000001BB 45
105    000001BC 43
105    000001BD 54
106    000001BE 1206          DC.W  GVR
107    000001C0 0000 0000     DC.L  ASM_INTERSECT
108                           REFR  ASM_MEMAVAIL
109    000001C4 0C            DC.B  12,'ASM_MEMAVAIL'
109    000001C5 41
109    000001C6 53
109    000001C7 4D
109    000001C8 5F
109    000001C9 4D
109    000001CA 45
109    000001CB 4D
109    000001CC 41
109    000001CD 56
109    000001CE 41
109    000001CF 49
109    000001D0 4C
110    000001D2 1206          DC.W  GVR
111    000001D4 0000 0000     DC.L  ASM_MEMAVAIL
112                           REFR  ASM_MOD
113    000001D8 07            DC.B  7,'ASM_MOD'
113    000001D9 41
113    000001DA 53
113    000001DB 4D
113    000001DC 5F
113    000001DD 4D
113    000001DE 4F
113    000001DF 44
114    000001E0 1206          DC.W  GVR
115    000001E2 0000 0000     DC.L  ASM_MOD
116                           REFR  ASM_MOVEL
117    000001E6 09            DC.B  9,'ASM_MOVEL'
117    000001E7 41
117    000001E8 53
```



```
117    000001E9 4D
117    000001EA 5F
117    000001EB 4D
117    000001EC 4F
117    000001ED 56
117    000001EE 45
117    000001EF 4C
118    000001F0 1206          DC.W  GVR
119    000001F2 0000 0000     DC.L  ASM_MOVEL
120                           REFR  ASM_MOVELEFT
121    000001F6 0C            DC.B  12,'ASM_MOVELEFT'
121    000001F7 41
121    000001F8 53
121    000001F9 4D
121    000001FA 5F
121    000001FB 4D
121    000001FC 4F
121    000001FD 56
121    000001FE 45
121    000001FF 4C
121    00000200 45
121    00000201 46
121    00000202 54
122    00000204 1206          DC.W  GVR
123    00000206 0000 0000     DC.L  ASM_MOVELEFT
124                           REFR  ASM_MOVER
125    0000020A 09            DC.B  9,'ASM_MOVER'
125    0000020B 41
125    0000020C 53
125    0000020D 4D
125    0000020E 5F
125    0000020F 4D
125    00000210 4F
125    00000211 56
125    00000212 45
125    00000213 52
126    00000214 1206          DC.W  GVR
127    00000216 0000 0000     DC.L  ASM_MOVER
128                           REFR  ASM_MOVERIGHT
129    0000021A 0D            DC.B  13,'ASM_MOVERIGHT'
129    0000021B 41
129    0000021C 53
129    0000021D 4D
129    0000021E 5F
129    0000021F 4D
129    00000220 4F
129    00000221 56
129    00000222 45
129    00000223 52
129    00000224 49
129    00000225 47
129    00000226 48
129    00000227 54
130    00000228 1206          DC.W  GVR
131    0000022A 0000 0000     DC.L  ASM_MOVERIGHT
132                           REFR  ASM_MPY
133    0000022E 07            DC.B  7,'ASM_MPY'
```

```
  133    0000022F 41
  133    00000230 53
  133    00000231 4D
  133    00000232 5F
  133    00000233 4D
  133    00000234 50
  133    00000235 59
  134    00000236 1206          DC.W  GVR
  135    00000238 0000 0000     DC.L  ASM_MPY
  136                           REFR  ASM_NEQUAL
  137    0000023C 0A            DC.B  10,'ASM_NEQUAL'
  137    0000023D 41
  137    0000023E 53
  137    0000023F 4D
  137    00000240 5F
  137    00000241 4E
  137    00000242 45
  137    00000243 51
  137    00000244 55
  137    00000245 41
  137    00000246 4C
  138    00000248 1206          DC.W  GVR
  139    0000024A 0000 0000     DC.L  ASM_NEQUAL
  140                           REFR  ASM_NEWBYTES
  141    0000024E 0C            DC.B  12,'ASM_NEWBYTES'
  141    0000024F 41
  141    00000250 53
  141    00000251 4D
  141    00000252 5F
  141    00000253 4E
  141    00000254 45
  141    00000255 57
  141    00000256 42
  141    00000257 59
  141    00000258 54
  141    00000259 45
  141    0000025A 53
  142    0000025C 1206          DC.W  GVR
  143    0000025E 0000 0000     DC.L  ASM_NEWBYTES
  144                           REFR  ASM_NEWWORDS
  145    00000262 0C            DC.B  12,'ASM_NEWWORDS'
  145    00000263 41
  145    00000264 53
  145    00000265 4D
  145    00000266 5F
  145    00000267 4E
  145    00000268 45
  145    00000269 57
  145    0000026A 57
  145    0000026B 4F
  145    0000026C 52
  145    0000026D 44
  145    0000026E 53
  146    00000270 1206          DC.W  GVR
  147    00000272 0000 0000     DC.L  ASM_NEWWORDS
  148                           REFR  ASM_POS
  149    00000276 07            DC.B  7,'ASM_POS'
```



```
  149    00000277 41
  149    00000278 53
  149    00000279 4D
  149    0000027A 5F
  149    0000027B 50
  149    0000027C 4F
  149    0000027D 53
  150    0000027E 1206          DC.W  GVR
  151    00000280 0000 0000     DC.L  ASM_POS
  152                           REFR  ASM_POWERUP
  153    00000284 0B            DC.B  11,'ASM_POWERUP'
  153    00000285 41
  153    00000286 53
  153    00000287 4D
  153    00000288 5F
  153    00000289 50
  153    0000028A 4F
  153    0000028B 57
  153    0000028C 45
  153    0000028D 52
  153    0000028E 55
  153    0000028F 50
  154    00000290 1206          DC.W  GVR
  155    00000292 0000 0000     DC.L  ASM_POWERUP
  156                           REFR  ASM_RMOVEL
  157    00000296 0A            DC.B  10,'ASM_RMOVEL'
  157    00000297 41
  157    00000298 53
  157    00000299 4D
  157    0000029A 5F
  157    0000029B 52
  157    0000029C 4D
  157    0000029D 4F
  157    0000029E 56
  157    0000029F 45
  157    000002A0 4C
  158    000002A2 1206          DC.W  GVR
  159    000002A4 0000 0000     DC.L  ASM_RMOVEL
  160                           REFR  ASM_RMOVER
  161    000002A8 0A            DC.B  10,'ASM_RMOVER'
  161    000002A9 41
  161    000002AA 53
  161    000002AB 4D
  161    000002AC 5F
  161    000002AD 52
  161    000002AE 4D
  161    000002AF 4F
  161    000002B0 56
  161    000002B1 45
  161    000002B2 52
  162    000002B4 1206          DC.W  GVR
  163    000002B6 0000 0000     DC.L  ASM_RMOVER
  164                           REFR  ASM_UNION
  165    000002BA 09            DC.B  9,'ASM_UNION'
  165    000002BB 41
  165    000002BC 53
  165    000002BD 4D
```

```
165     000002BE 5F
165     000002BF 55
165     000002C0 4E
165     000002C1 49
165     000002C2 4F
165     000002C3 4E
166     000002C4 1206           DC.W  GVR
167     000002C6 0000 0000      DC.L  ASM_UNION
168                             REFR  ASM_USERPROGRAM
169     000002CA 0F             DC.B  15,'ASM_USERPROGRAM'
169     000002CB 41
169     000002CC 53
169     000002CD 4D
169     000002CE 5F
169     000002CF 55
169     000002D0 53
169     000002D1 45
169     000002D2 52
169     000002D3 50
169     000002D4 52
169     000002D5 4F
169     000002D6 47
169     000002D7 52
169     000002D8 41
169     000002D9 4D
170     000002DA 1206           DC.W  GVR
171     000002DC 0000 0000      DC.L  ASM_USERPROGRAM
172                             REFR  BOOTDAMMODULE
173     000002E0 0D             DC.B  13,'BOOTDAMMODULE'
173     000002E1 42
173     000002E2 4F
173     000002E3 4F
173     000002E4 54
173     000002E5 44
173     000002E6 41
173     000002E7 4D
173     000002E8 4D
173     000002E9 4F
173     000002EA 44
173     000002EB 55
173     000002EC 4C
173     000002ED 45
174     000002EE 1206           DC.W  GVR
175     000002F0 0000 0000      DC.L  BOOTDAMMODULE
176                             REFR  BOOTDAMMODULE_BOOTDAM
177     000002F4 15             DC.B  21,'BOOTDAMMODULE_BOOTDAM'
177     000002F5 42
177     000002F6 4F
177     000002F7 4F
177     000002F8 54
177     000002F9 44
177     000002FA 41
177     000002FB 4D
177     000002FC 4D
177     000002FD 4F
177     000002FE 44
177     000002FF 55
```



```
177     00000300 4C
177     00000301 45
177     00000302 5F
177     00000303 42
177     00000304 4F
177     00000305 4F
177     00000306 54
177     00000307 44
177     00000308 41
177     00000309 4D
178     0000030A 1206           DC.W  GVR
179     0000030C 0000 0000      DC.L  BOOTDAMMODULE_BOOTDAM
180                             REFR  BOOTDAMMODULE_BOOTDAMMODULE
181     00000310 1B             DC.B  27,'BOOTDAMMODULE_BOOTDAMMODULE'
181     00000311 42
181     00000312 4F
181     00000313 4F
181     00000314 54
181     00000315 44
181     00000316 41
181     00000317 4D
181     00000318 4D
181     00000319 4F
181     0000031A 44
181     0000031B 55
181     0000031C 4C
181     0000031D 45
181     0000031E 5F
181     0000031F 42
181     00000320 4F
181     00000321 4F
181     00000322 54
181     00000323 44
181     00000324 41
181     00000325 4D
181     00000326 4D
181     00000327 4F
181     00000328 44
181     00000329 55
181     0000032A 4C
181     0000032B 45
182     0000032C 1206           DC.W  GVR
183     0000032E 0000 0000      DC.L  BOOTDAMMODULE_BOOTDAMMODULE
184                             REFR  BOOTDAMMODULE_BOOTNODE
185     00000332 16             DC.B  22,'BOOTDAMMODULE_BOOTNODE'
185     00000333 42
185     00000334 4F
185     00000335 4F
185     00000336 54
185     00000337 44
185     00000338 41
185     00000339 4D
185     0000033A 4D
185     0000033B 4F
185     0000033C 44
185     0000033D 55
185     0000033E 4C
```

```
185     0000033F 45
185     00000340 5F
185     00000341 42
185     00000342 4F
185     00000343 4F
185     00000344 54
185     00000345 4E
185     00000346 4F
185     00000347 44
185     00000348 45
186     0000034A 1206           DC.W  GVR
187     0000034C 0000 0000      DC.L  BOOTDAMMODULE_BOOTNODE
188                             REFR  BOOTDAMMODULE_BOOTTM
189     00000350 14             DC.B  20,'BOOTDAMMODULE_BOOTTM'
189     00000351 42
189     00000352 4F
189     00000353 4F
189     00000354 54
189     00000355 44
189     00000356 41
189     00000357 4D
189     00000358 4D
189     00000359 4F
189     0000035A 44
189     0000035B 55
189     0000035C 4C
189     0000035D 45
189     0000035E 5F
189     0000035F 42
189     00000360 4F
189     00000361 4F
189     00000362 54
189     00000363 54
189     00000364 4D
190     00000366 1206           DC.W  GVR
191     00000368 0000 0000      DC.L  BOOTDAMMODULE_BOOTTM
192                             REFR  BOOTDAMMODULE_INITBOOTDAM
193     0000036C 19             DC.B  25,'BOOTDAMMODULE_INITBOOTDAM'
193     0000036D 42
193     0000036E 4F
193     0000036F 4F
193     00000370 54
193     00000371 44
193     00000372 41
193     00000373 4D
193     00000374 4D
193     00000375 4F
193     00000376 44
193     00000377 55
193     00000378 4C
193     00000379 45
193     0000037A 5F
193     0000037B 49
193     0000037C 4E
193     0000037D 49
193     0000037E 54
193     0000037F 42
```



```
193     00000380 4F
193     00000381 4F
193     00000382 54
193     00000383 44
193     00000384 41
193     00000385 4D
194     00000386 1206           DC.W  GVR
195     00000388 0000 0000      DC.L  BOOTDAMMODULE_INITBOOTDAM
196                             REFR  BOOTDAMMODULE_SRMNODE
197     0000038C 15             DC.B  21,'BOOTDAMMODULE_SRMNODE'
197     0000038D 42
197     0000038E 4F
197     0000038F 4F
197     00000390 54
197     00000391 44
197     00000392 41
197     00000393 4D
197     00000394 4D
197     00000395 4F
197     00000396 44
197     00000397 55
197     00000398 4C
197     00000399 45
197     0000039A 5F
197     0000039B 53
197     0000039C 52
197     0000039D 4D
197     0000039E 4E
197     0000039F 4F
197     000003A0 44
197     000003A1 45
198     000003A2 1206           DC.W  GVR
199     000003A4 0000 0000      DC.L  BOOTDAMMODULE_SRMNODE
200                             REFR  BOOTDAMMODULE__BASE
201     000003A8 13             DC.B  19,'BOOTDAMMODULE__BASE'
201     000003A9 42
201     000003AA 4F
201     000003AB 4F
201     000003AC 54
201     000003AD 44
201     000003AE 41
201     000003AF 4D
201     000003B0 4D
201     000003B1 4F
201     000003B2 44
201     000003B3 55
201     000003B4 4C
201     000003B5 45
201     000003B6 5F
201     000003B7 5F
201     000003B8 42
201     000003B9 41
201     000003BA 53
201     000003BB 45
202     000003BC 1206           DC.W  GVR
203     000003BE 0000 0000      DC.L  BOOTDAMMODULE__BASE
204                             REFR  BOOT_FINDFILE
```

```
  205    000003C2 0D           DC.B  13,'BOOT_FINDFILE'
  205    000003C3 42
  205    000003C4 4F
  205    000003C5 4F
  205    000003C6 54
  205    000003C7 5F
  205    000003C8 46
  205    000003C9 49
  205    000003CA 4E
  205    000003CB 44
  205    000003CC 46
  205    000003CD 49
  205    000003CE 4C
  205    000003CF 45
  206    000003D0 1206         DC.W  GVR
  207    000003D2 0000 0000    DC.L  BOOT_FINDFILE
  208                          REFR  BOOT_LIFHEAD
  209    000003D6 0C           DC.B  12,'BOOT_LIFHEAD'
  209    000003D7 42
  209    000003D8 4F
  209    000003D9 4F
  209    000003DA 54
  209    000003DB 5F
  209    000003DC 4C
  209    000003DD 49
  209    000003DE 46
  209    000003DF 48
  209    000003E0 45
  209    000003E1 41
  209    000003E2 44
  210    000003E4 1206         DC.W  GVR
  211    000003E6 0000 0000    DC.L  BOOT_LIFHEAD
  212                          REFR  BOOT_MFCLOSE
  213    000003EA 0C           DC.B  12,'BOOT_MFCLOSE'
  213    000003EB 42
  213    000003EC 4F
  213    000003ED 4F
  213    000003EE 54
  213    000003EF 5F
  213    000003F0 4D
  213    000003F1 46
  213    000003F2 43
  213    000003F3 4C
  213    000003F4 4F
  213    000003F5 53
  213    000003F6 45
  214    000003F8 1206         DC.W  GVR
  215    000003FA 0000 0000    DC.L  BOOT_MFCLOSE
  216                          REFR  BOOT_MFOPEN
  217    000003FE 0B           DC.B  11,'BOOT_MFOPEN'
  217    000003FF 42
  217    00000400 4F
  217    00000401 4F
  217    00000402 54
  217    00000403 5F
  217    00000404 4D
  217    00000405 46
```



```
  217    00000406 4F
  217    00000407 50
  217    00000408 45
  217    00000409 4E
  218    0000040A 1206         DC.W  GVR
  219    0000040C 0000 0000    DC.L  BOOT_MFOPEN
  220                          REFR  BOOT_MINIT
  221    00000410 0A           DC.B  10,'BOOT_MINIT'
  221    00000411 42
  221    00000412 4F
  221    00000413 4F
  221    00000414 54
  221    00000415 5F
  221    00000416 4D
  221    00000417 49
  221    00000418 4E
  221    00000419 49
  221    0000041A 54
  222    0000041C 1206         DC.W  GVR
  223    0000041E 0000 0000    DC.L  BOOT_MINIT
  224                          REFR  BOOT_MREAD
  225    00000422 0A           DC.B  10,'BOOT_MREAD'
  225    00000423 42
  225    00000424 4F
  225    00000425 4F
  225    00000426 54
  225    00000427 5F
  225    00000428 4D
  225    00000429 52
  225    0000042A 45
  225    0000042B 41
  225    0000042C 44
  226    0000042E 1206         DC.W  GVR
  227    00000430 0000 0000    DC.L  BOOT_MREAD
  228                          REFR  EVALGVR
  229    00000434 07           DC.B  7,'EVALGVR'
  229    00000435 45
  229    00000436 56
  229    00000437 41
  229    00000438 4C
  229    00000439 47
  229    0000043A 56
  229    0000043B 52
  230    0000043C 1206         DC.W  GVR
  231    0000043E 0000 0000    DC.L  EVALGVR
  232                          REFR  FS_FWRITESTRINT
  233    00000442 0F           DC.B  15,'FS_FWRITESTRINT'
  233    00000443 46
  233    00000444 53
  233    00000445 5F
  233    00000446 46
  233    00000447 57
  233    00000448 52
  233    00000449 49
  233    0000044A 54
  233    0000044B 45
  233    0000044C 53
```

```
  233    0000044D 54
  233    0000044E 52
  233    0000044F 49
  233    00000450 4E
  233    00000451 54
  234    00000452 1206          DC.W  GVR
  235    00000454 0000 0000     DC.L  FS_FWRITESTRINT
  236                           REFR  G_DOLLAR
  237    00000458 08            DC.B  8,'G_DOLLAR'
  237    00000459 47
  237    0000045A 5F
  237    0000045B 44
  237    0000045C 4F
  237    0000045D 4C
  237    0000045E 4C
  237    0000045F 41
  237    00000460 52
  238    00000462 1206          DC.W  GVR
  239    00000464 0000 0000     DC.L  G_DOLLAR
  240                           REFR  INITLOAD
  241    00000468 08            DC.B  8,'INITLOAD'
  241    00000469 49
  241    0000046A 4E
  241    0000046B 49
  241    0000046C 54
  241    0000046D 4C
  241    0000046E 4F
  241    0000046F 41
  241    00000470 44
  242    00000472 1206          DC.W  GVR
  243    00000474 0000 0000     DC.L  INITLOAD
  244                           REFR  INITLOAD_INITLOAD
  245    00000478 11            DC.B  17,'INITLOAD_INITLOAD'
  245    00000479 49
  245    0000047A 4E
  245    0000047B 49
  245    0000047C 54
  245    0000047D 4C
  245    0000047E 4F
  245    0000047F 41
  245    00000480 44
  245    00000481 5F
  245    00000482 49
  245    00000483 4E
  245    00000484 49
  245    00000485 54
  245    00000486 4C
  245    00000487 4F
  245    00000488 41
  245    00000489 44
  246    0000048A 1206          DC.W  GVR
  247    0000048C 0000 0000     DC.L  INITLOAD_INITLOAD
  248                           REFR  INITLOAD_SYSPREFIX
  249    00000490 12            DC.B  18,'INITLOAD_SYSPREFIX'
  249    00000491 49
  249    00000492 4E
  249    00000493 49
```



```
  249    00000494 54
  249    00000495 4C
  249    00000496 4F
  249    00000497 41
  249    00000498 44
  249    00000499 5F
  249    0000049A 53
  249    0000049B 59
  249    0000049C 53
  249    0000049D 50
  249    0000049E 52
  249    0000049F 45
  249    000004A0 46
  249    000004A1 49
  249    000004A2 58
  250    000004A4 1206          DC.W  GVR
  251    000004A6 0000 0000     DC.L  INITLOAD_SYSPREFIX
  252                           REFR  INITLOAD__BASE
  253    000004AA 0E            DC.B  14,'INITLOAD__BASE'
  253    000004AB 49
  253    000004AC 4E
  253    000004AD 49
  253    000004AE 54
  253    000004AF 4C
  253    000004B0 4F
  253    000004B1 41
  253    000004B2 44
  253    000004B3 5F
  253    000004B4 5F
  253    000004B5 42
  253    000004B6 41
  253    000004B7 53
  253    000004B8 45
  254    000004BA 1206          DC.W  GVR
  255    000004BC 0000 0000     DC.L  INITLOAD__BASE
  256                           REFR  LOADER
  257    000004C0 06            DC.B  6,'LOADER'
  257    000004C1 4C
  257    000004C2 4F
  257    000004C3 41
  257    000004C4 44
  257    000004C5 45
  257    000004C6 52
  258    000004C8 1206          DC.W  GVR
  259    000004CA 0000 0000     DC.L  LOADER
  260                           REFR  LOADER_CHECKREV
  261    000004CE 0F            DC.B  15,'LOADER_CHECKREV'
  261    000004CF 4C
  261    000004D0 4F
  261    000004D1 41
  261    000004D2 44
  261    000004D3 45
  261    000004D4 52
  261    000004D5 5F
  261    000004D6 43
  261    000004D7 48
  261    000004D8 45
```

```
  261    000004D9 43
  261    000004DA 4B
  261    000004DB 52
  261    000004DC 45
  261    000004DD 56
  262    000004DE 1206          DC.W  GVR
  263    000004E0 0000 0000     DC.L  LOADER_CHECKREV
  264                           REFR  LOADER_CLOSEFILES
  265    000004E4 11            DC.B  17,'LOADER_CLOSEFILES'
  265    000004E5 4C
  265    000004E6 4F
  265    000004E7 41
  265    000004E8 44
  265    000004E9 45
  265    000004EA 52
  265    000004EB 5F
  265    000004EC 43
  265    000004ED 4C
  265    000004EE 4F
  265    000004EF 53
  265    000004F0 45
  265    000004F1 46
  265    000004F2 49
  265    000004F3 4C
  265    000004F4 45
  265    000004F5 53
  266    000004F6 1206          DC.W  GVR
  267    000004F8 0000 0000     DC.L  LOADER_CLOSEFILES
  268                           REFR  LOADER_COUNTCODE
  269    000004FC 10            DC.B  16,'LOADER_COUNTCODE'
  269    000004FD 4C
  269    000004FE 4F
  269    000004FF 41
  269    00000500 44
  269    00000501 45
  269    00000502 52
  269    00000503 5F
  269    00000504 43
  269    00000505 4F
  269    00000506 55
  269    00000507 4E
  269    00000508 54
  269    00000509 43
  269    0000050A 4F
  269    0000050B 44
  269    0000050C 45
  270    0000050E 1206          DC.W  GVR
  271    00000510 0000 0000     DC.L  LOADER_COUNTCODE
  272                           REFR  LOADER_GETBYTES
  273    00000514 0F            DC.B  15,'LOADER_GETBYTES'
  273    00000515 4C
  273    00000516 4F
  273    00000517 41
  273    00000518 44
  273    00000519 45
  273    0000051A 52
  273    0000051B 5F
```



```
  273    0000051C 47
  273    0000051D 45
  273    0000051E 54
  273    0000051F 42
  273    00000520 59
  273    00000521 54
  273    00000522 45
  273    00000523 53
  274    00000524 1206          DC.W  GVR
  275    00000526 0000 0000     DC.L  LOADER_GETBYTES
  276                           REFR  LOADER_INITLOADER
  277    0000052A 11            DC.B  17,'LOADER_INITLOADER'
  277    0000052B 4C
  277    0000052C 4F
  277    0000052D 41
  277    0000052E 44
  277    0000052F 45
  277    00000530 52
  277    00000531 5F
  277    00000532 49
  277    00000533 4E
  277    00000534 49
  277    00000535 54
  277    00000536 4C
  277    00000537 4F
  277    00000538 41
  277    00000539 44
  277    0000053A 45
  277    0000053B 52
  278    0000053C 1206          DC.W  GVR
  279    0000053E 0000 0000     DC.L  LOADER_INITLOADER
  280                           REFR  LOADER_LOADER
  281    00000542 0D            DC.B  13,'LOADER_LOADER'
  281    00000543 4C
  281    00000544 4F
  281    00000545 41
  281    00000546 44
  281    00000547 45
  281    00000548 52
  281    00000549 5F
  281    0000054A 4C
  281    0000054B 4F
  281    0000054C 41
  281    0000054D 44
  281    0000054E 45
  281    0000054F 52
  282    00000550 1206          DC.W  GVR
  283    00000552 0000 0000     DC.L  LOADER_LOADER
  284                           REFR  LOADER_LOADINFO
  285    00000556 0F            DC.B  15,'LOADER_LOADINFO'
  285    00000557 4C
  285    00000558 4F
  285    00000559 41
  285    0000055A 44
  285    0000055B 45
  285    0000055C 52
  285    0000055D 5F
```

```
  285    0000055E 4C
  285    0000055F 4F
  285    00000560 41
  285    00000561 44
  285    00000562 49
  285    00000563 4E
  285    00000564 46
  285    00000565 4F
  286    00000566 1206          DC.W  GVR
  287    00000568 0000 0000     DC.L  LOADER_LOADINFO
  288                           REFR  LOADER_LOADQ
  289    0000056C 0C            DC.B  12,'LOADER_LOADQ'
  289    0000056D 4C
  289    0000056E 4F
  289    0000056F 41
  289    00000570 44
  289    00000571 45
  289    00000572 52
  289    00000573 5F
  289    00000574 4C
  289    00000575 4F
  289    00000576 41
  289    00000577 44
  289    00000578 51
  290    0000057A 1206          DC.W  GVR
  291    0000057C 0000 0000     DC.L  LOADER_LOADQ
  292                           REFR  LOADER_LOADTEXT
  293    00000580 0F            DC.B  15,'LOADER_LOADTEXT'
  293    00000581 4C
  293    00000582 4F
  293    00000583 41
  293    00000584 44
  293    00000585 45
  293    00000586 52
  293    00000587 5F
  293    00000588 4C
  293    00000589 4F
  293    0000058A 41
  293    0000058B 44
  293    0000058C 54
  293    0000058D 45
  293    0000058E 58
  293    0000058F 54
  294    00000590 1206          DC.W  GVR
  295    00000592 0000 0000     DC.L  LOADER_LOADTEXT
  296                           REFR  LOADER_MARKUSER
  297    00000596 0F            DC.B  15,'LOADER_MARKUSER'
  297    00000597 4C
  297    00000598 4F
  297    00000599 41
  297    0000059A 44
  297    0000059B 45
  297    0000059C 52
  297    0000059D 5F
  297    0000059E 4D
  297    0000059F 41
  297    000005A0 52
```



```
  297    000005A1 4B
  297    000005A2 55
  297    000005A3 53
  297    000005A4 45
  297    000005A5 52
  298    000005A6 1206          DC.W  GVR
  299    000005A8 0000 0000     DC.L  LOADER_MARKUSER
  300                           REFR  LOADER_MATCHFILE
  301    000005AC 10            DC.B  16,'LOADER_MATCHFILE'
  301    000005AD 4C
  301    000005AE 4F
  301    000005AF 41
  301    000005B0 44
  301    000005B1 45
  301    000005B2 52
  301    000005B3 5F
  301    000005B4 4D
  301    000005B5 41
  301    000005B6 54
  301    000005B7 43
  301    000005B8 48
  301    000005B9 46
  301    000005BA 49
  301    000005BB 4C
  301    000005BC 45
  302    000005BE 1206          DC.W  GVR
  303    000005C0 0000 0000     DC.L  LOADER_MATCHFILE
  304                           REFR  LOADER_MOVEDEFS
  305    000005C4 0F            DC.B  15,'LOADER_MOVEDEFS'
  305    000005C5 4C
  305    000005C6 4F
  305    000005C7 41
  305    000005C8 44
  305    000005C9 45
  305    000005CA 52
  305    000005CB 5F
  305    000005CC 4D
  305    000005CD 4F
  305    000005CE 56
  305    000005CF 45
  305    000005D0 44
  305    000005D1 45
  305    000005D2 46
  305    000005D3 53
  306    000005D4 1206          DC.W  GVR
  307    000005D6 0000 0000     DC.L  LOADER_MOVEDEFS
  308                           REFR  LOADER_OPENLINKF
  309    000005DA 10            DC.B  16,'LOADER_OPENLINKF'
  309    000005DB 4C
  309    000005DC 4F
  309    000005DD 41
  309    000005DE 44
  309    000005DF 45
  309    000005E0 52
  309    000005E1 5F
  309    000005E2 4F
  309    000005E3 50
```

```
309     000005E4 45
309     000005E5 4E
309     000005E6 4C
309     000005E7 49
309     000005E8 4E
309     000005E9 4B
309     000005EA 46
310     000005EC 1206           DC.W  GVR
311     000005EE 0000 0000      DC.L  LOADER_OPENLINKF
312                             REFR  LOADER_RELEASEUSER
313     000005F2 12             DC.B  18,'LOADER_RELEASEUSER'
313     000005F3 4C
313     000005F4 4F
313     000005F5 41
313     000005F6 44
313     000005F7 45
313     000005F8 52
313     000005F9 5F
313     000005FA 52
313     000005FB 45
313     000005FC 4C
313     000005FD 45
313     000005FE 41
313     000005FF 53
313     00000600 45
313     00000601 55
313     00000602 53
313     00000603 45
313     00000604 52
314     00000606 1206           DC.W  GVR
315     00000608 0000 0000      DC.L  LOADER_RELEASEUSER
316                             REFR  LOADER_ZEROMEM
317     0000060C 0E             DC.B  14,'LOADER_ZEROMEM'
317     0000060D 4C
317     0000060E 4F
317     0000060F 41
317     00000610 44
317     00000611 45
317     00000612 52
317     00000613 5F
317     00000614 5A
317     00000615 45
317     00000616 52
317     00000617 4F
317     00000618 4D
317     00000619 45
317     0000061A 4D
318     0000061C 1206           DC.W  GVR
319     0000061E 0000 0000      DC.L  LOADER_ZEROMEM
320                             REFR  LOADER__BASE
321     00000622 0C             DC.B  12,'LOADER__BASE'
321     00000623 4C
321     00000624 4F
321     00000625 41
321     00000626 44
321     00000627 45
321     00000628 52
```



```
321     00000629 5F
321     0000062A 5F
321     0000062B 42
321     0000062C 41
321     0000062D 53
321     0000062E 45
322     00000630 1206           DC.W  GVR
323     00000632 0000 0000      DC.L  LOADER__BASE
324                             REFR  M68KTYPE
325     00000636 08             DC.B  8,'M68KTYPE'
325     00000637 4D
325     00000638 36
325     00000639 38
325     0000063A 4B
325     0000063B 54
325     0000063C 59
325     0000063D 50
325     0000063E 45
326     00000640 1206           DC.W  GVR
327     00000642 0000 0000      DC.L  M68KTYPE
328                             REFR  MATCHDEFEXT
329     00000646 0B             DC.B  11,'MATCHDEFEXT'
329     00000647 4D
329     00000648 41
329     00000649 54
329     0000064A 43
329     0000064B 48
329     0000064C 44
329     0000064D 45
329     0000064E 46
329     0000064F 45
329     00000650 58
329     00000651 54
330     00000652 1206           DC.W  GVR
331     00000654 0000 0000      DC.L  MATCHDEFEXT
332                             REFR  MINI
333     00000658 04             DC.B  4,'MINI'
333     00000659 4D
333     0000065A 49
333     0000065B 4E
333     0000065C 49
334     0000065E 1206           DC.W  GVR
335     00000660 0000 0000      DC.L  MINI
336                             REFR  MINI_IORESC
337     00000664 0B             DC.B  11,'MINI_IORESC'
337     00000665 4D
337     00000666 49
337     00000667 4E
337     00000668 49
337     00000669 5F
337     0000066A 49
337     0000066B 4F
337     0000066C 52
337     0000066D 45
337     0000066E 53
337     0000066F 43
338     00000670 1206           DC.W  GVR
```

```
  339    00000672 0000 0000    DC.L  MINI_IORESC
  340                          REFR  MINI_IORVAL
  341    00000676 0B           DC.B  11,'MINI_IORVAL'
  341    00000677 4D
  341    00000678 49
  341    00000679 4E
  341    0000067A 49
  341    0000067B 5F
  341    0000067C 49
  341    0000067D 4F
  341    0000067E 52
  341    0000067F 56
  341    00000680 41
  341    00000681 4C
  342    00000682 1206         DC.W  GVR
  343    00000684 0000 0000    DC.L  MINI_IORVAL
  344                          REFR  MINI_MINI
  345    00000688 09           DC.B  9,'MINI_MINI'
  345    00000689 4D
  345    0000068A 49
  345    0000068B 4E
  345    0000068C 49
  345    0000068D 5F
  345    0000068E 4D
  345    0000068F 49
  345    00000690 4E
  345    00000691 49
  346    00000692 1206         DC.W  GVR
  347    00000694 0000 0000    DC.L  MINI_MINI
  348                          REFR  MINI_MINIIO
  349    00000698 0B           DC.B  11,'MINI_MINIIO'
  349    00000699 4D
  349    0000069A 49
  349    0000069B 4E
  349    0000069C 49
  349    0000069D 5F
  349    0000069E 4D
  349    0000069F 49
  349    000006A0 4E
  349    000006A1 49
  349    000006A2 49
  349    000006A3 4F
  350    000006A4 1206         DC.W  GVR
  351    000006A6 0000 0000    DC.L  MINI_MINIIO
  352                          REFR  MINI__BASE
  353    000006AA 0A           DC.B  10,'MINI__BASE'
  353    000006AB 4D
  353    000006AC 49
  353    000006AD 4E
  353    000006AE 49
  353    000006AF 5F
  353    000006B0 5F
  353    000006B1 42
  353    000006B2 41
  353    000006B3 53
  353    000006B4 45
  354    000006B6 1206         DC.W  GVR
```



```
  355    000006B8 0000 0000    DC.L  MINI__BASE
  356                          REFR  MSYSFLAGS
  357    000006BC 09           DC.B  9,'MSYSFLAGS'
  357    000006BD 4D
  357    000006BE 53
  357    000006BF 59
  357    000006C0 53
  357    000006C1 46
  357    000006C2 4C
  357    000006C3 41
  357    000006C4 47
  357    000006C5 53
  358    000006C6 1206         DC.W  GVR
  359    000006C8 0000 0000    DC.L  MSYSFLAGS
  360                          REFR  RELOCATE
  361    000006CC 08           DC.B  8,'RELOCATE'
  361    000006CD 52
  361    000006CE 45
  361    000006CF 4C
  361    000006D0 4F
  361    000006D1 43
  361    000006D2 41
  361    000006D3 54
  361    000006D4 45
  362    000006D6 1206         DC.W  GVR
  363    000006D8 0000 0000    DC.L  RELOCATE
  364                          REFR  STACKFUDGE
  365    000006DC 0A           DC.B  10,'STACKFUDGE'
  365    000006DD 53
  365    000006DE 54
  365    000006DF 41
  365    000006E0 43
  365    000006E1 4B
  365    000006E2 46
  365    000006E3 55
  365    000006E4 44
  365    000006E5 47
  365    000006E6 45
  366    000006E8 1206         DC.W  GVR
  367    000006EA 0000 0000    DC.L  STACKFUDGE
  368                          REFR  SYSGLOBALS
  369    000006EE 0A           DC.B  10,'SYSGLOBALS'
  369    000006EF 53
  369    000006F0 59
  369    000006F1 53
  369    000006F2 47
  369    000006F3 4C
  369    000006F4 4F
  369    000006F5 42
  369    000006F6 41
  369    000006F7 4C
  369    000006F8 53
  370    000006FA 1206         DC.W  GVR
  371    000006FC 0000 0000    DC.L  SYSGLOBALS
  372                          REFR  SYSGLOBALS_FSIDC
  373    00000700 10           DC.B  16,'SYSGLOBALS_FSIDC'
  373    00000701 53
```

```
  373    00000702 59
  373    00000703 53
  373    00000704 47
  373    00000705 4C
  373    00000706 4F
  373    00000707 42
  373    00000708 41
  373    00000709 4C
  373    0000070A 53
  373    0000070B 5F
  373    0000070C 46
  373    0000070D 53
  373    0000070E 49
  373    0000070F 44
  373    00000710 43
  374    00000712 1206          DC.W  GVR
  375    00000714 0000 0000     DC.L  SYSGLOBALS_FSIDC
  376                           REFR  SYSGLOBALS_SYSGLOBALS
  377    00000718 15            DC.B  21,'SYSGLOBALS_SYSGLOBALS'
  377    00000719 53
  377    0000071A 59
  377    0000071B 53
  377    0000071C 47
  377    0000071D 4C
  377    0000071E 4F
  377    0000071F 42
  377    00000720 41
  377    00000721 4C
  377    00000722 53
  377    00000723 5F
  377    00000724 53
  377    00000725 59
  377    00000726 53
  377    00000727 47
  377    00000728 4C
  377    00000729 4F
  377    0000072A 42
  377    0000072B 41
  377    0000072C 4C
  377    0000072D 53
  378    0000072E 1206          DC.W  GVR
  379    00000730 0000 0000     DC.L  SYSGLOBALS_SYSGLOBALS
  380                           REFR  SYSGLOBALS__BASE
  381    00000734 10            DC.B  16,'SYSGLOBALS__BASE'
  381    00000735 53
  381    00000736 59
  381    00000737 53
  381    00000738 47
  381    00000739 4C
  381    0000073A 4F
  381    0000073B 42
  381    0000073C 41
  381    0000073D 4C
  381    0000073E 53
  381    0000073F 5F
  381    00000740 5F
  381    00000741 42
```



```
  381    00000742 41
  381    00000743 53
  381    00000744 45
  382    00000746 1206          DC.W  GVR
  383    00000748 0000 0000     DC.L  SYSGLOBALS__BASE
  384
  385             0000 074C endd    equ *
  386
  387                               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_CACHE_OFF	REL	32	0000000A
ASM_CACHE_ON	REL	36	0000000E
ASM_CI_SWITCH	REL	40	00000012
ASM_CPYMSG	REL	44	00000016
ASM_DIFFERENCE	REL	48	00000019
ASM_DIV	REL	52	0000001D
ASM_EQUAL	REL	56	0000001F
ASM_ERRMSG	REL	60	00000022
ASM_FASTMOVE	REL	64	00000025
ASM_FINDROMS	REL	68	00000029
ASM_FLPYINIT	REL	72	0000002D
ASM_FLPYMREAD	REL	76	00000031
ASM_FLPYMWRITE	REL	80	00000035
ASM_FLPYREAD	REL	84	00000039
ASM_FLPY_WRT	REL	88	0000003D
ASM_F_PWR_ON	REL	92	00000041
ASM_IN	REL	96	00000045
ASM_INCLUSION	REL	100	00000047
ASM_INTERSECT	REL	104	0000004B
ASM_MEMAVAIL	REL	108	0000004F
ASM_MOD	REL	112	00000053
ASM_MOVEL	REL	116	00000055
ASM_MOVELEFT	REL	120	00000058
ASM_MOVER	REL	124	0000005C
ASM_MOVERIGHT	REL	128	0000005F
ASM_MPY	REL	132	00000063
ASM_NEQUAL	REL	136	00000065
ASM_NEWBYTES	REL	140	00000068
ASM_NEWWORDS	REL	144	0000006C
ASM_POS	REL	148	00000070
ASM_POWERUP	REL	152	00000072
ASM_RMOVEL	REL	156	00000075
ASM_RMOVER	REL	160	00000078
ASM_UNION	REL	164	0000007B
ASM_USERPROGRAM	REL	168	0000007E
BOOTDAMMODULE	REL	172	00000082
BOOTDAMMODULE_BOOTDAM	REL	176	00000086
BOOTDAMMODULE_BOOTDAMMODULE	REL	180	0000008C
BOOTDAMMODULE_BOOTNODE	REL	184	00000093
BOOTDAMMODULE_BOOTTN	REL	188	00000099
BOOTDAMMODULE_INITBOOTDAM	REL	192	0000009F
BOOTDAMMODULE_SRMNODE	REL	196	000000A6
BOOTDAMMODULE__BASE	REL	200	000000AC
BOOT_FINDFILE	REL	204	000000B1
BOOT_LIFHEAD	REL	208	000000B5
BOOT_MFCLOSE	REL	212	000000B9
BOOT_MFOPEN	REL	216	000000BD
BOOT_MINIT	REL	220	000000C0

BOOT_MREAD	REL	224	000000C3
EVALGVR	REL	228	000000C6
FS_FWRITESTRINT	REL	232	000000C8
G_DOLLAR	REL	236	000000CC
INITLOAD	REL	240	000000CF
INITLOAD_INITLOAD	REL	244	000000D2
INITLOAD_SYSPREFIX	REL	248	000000D7
INITLOAD__BASE	REL	252	000000DC
LOADER	REL	256	000000E0
LOADER_CHECKREV	REL	260	000000E2
LOADER_CLOSEFILES	REL	264	000000E6
LOADER_COUNTCODE	REL	268	000000EB
LOADER_GETBYTES	REL	272	000000F0
LOADER_INITLOADER	REL	276	000000F4
LOADER_LOADER	REL	280	000000F9
LOADER_LOADINFO	REL	284	000000FD
LOADER_LOADQ	REL	288	00000101
LOADER_LOADTEXT	REL	292	00000105
LOADER_MARKUSER	REL	296	00000109
LOADER_MATCHFILE	REL	300	0000010D
LOADER_MOVEDEFS	REL	304	00000112
LOADER_OPENLINKF	REL	308	00000116
LOADER_RELEASEUSER	REL	312	0000011B
LOADER_ZEROMEM	REL	316	00000120
LOADER__BASE	REL	320	00000124
M68KTYPE	REL	324	00000128
MATCHDEFEXT	REL	328	0000012B
MINI	REL	332	0000012E
MINI_IORESC	REL	336	00000130
MINI_IORVAL	REL	340	00000133
MINI_MINI	REL	344	00000136
MINI_MINIIO	REL	348	00000139
MINI__BASE	REL	352	0000013C
MSYSFLAGS	REL	356	0000013F
RELOCATE	REL	360	00000142
STACKFUDGE	REL	364	00000145
SYSGLOBALS	REL	368	00000148
SYSGLOBALS_FSIDC	REL	372	0000014B
SYSGLOBALS_SYSGLOBALS	REL	376	00000150
SYSGLOBALS__BASE	REL	380	00000156

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
DFC             STREG      0                00000008
ENDD            REL      385                0000074C
GVR             ABS       15                00001206
SFC             STREG      0                00000009
SP              AREG       0                00000007
SR              STREG      0                00000006
SYSDEFTABLE     REL        5                00000000
USP             STREG      0                00000007
VBR             STREG      0                0000000A
X               REL       13                00000025
```

Part No. 98615-90074
E 0285
Microfiche No. 98615-99074

Printed in U.S.A.
Edition 1
February 1985