```
   1    Compilation of: INITDIR
   2
   3    D: Disk Print
   4    L: List Source Program
   5
   6    %include 'diomod.dcl';
   7    %include 'initdira.dcl';
   8       No Error(s) in Pass 1
   9
  10       No Error(s) in Pass 2
  11
  12        1 a 0000 initdir: procedure options(main);
  13        2 c 0000
  14        3 c 0000 /* REVISION HISTORY:
  15        4 c 0000    1/24/83 whf  converted to run on CCP/M-86
  16        5 c 0000   11/12/82 pb   fixed bug in clearout, buildnew, and reconstruction */
  17        6 c 0000
  18        7 c 0000 declare
  19        8 c 0006     COPYRIGHT char(44) static initial
  20        9 c 0006     ('COPYRIGHT (c) 1983 BY DIGITAL RESEARCH INC.');
  21       10 c 0006
  22       11 c 0006 /*
  23       12 c 0006             copyright(c) 1982, 1983
  24       13 c 0006             digital research
  25       14 c 0006             box 579
  26       15 c 0006             pacific grove, ca
  27       16 c 0006             93950
  28       17 c 0006   */
  29       18 c 0006
  30       19 c 0006 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  31       20 c 0006
  32       21 c 0006
  33       22 c 0006                    * * *  DISK INTERFACE * * *
  34       23 c 0006
  35       24 c 0006
  36       25 c 0006  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
  37       26 c 0006
  38       27+c 0006
  39       28+c 0006     dcl
  40       29+c 0006         memptr entry                  returns (ptr),
  41       30+c 0006         memsiz entry                  returns (fixed(15)),
  42       31+c 0006         memwds entry                  returns (fixed(15)),
  43       32+c 0006         dfcb0  entry                  returns (ptr),
  44       33+c 0006         dfcb1  entry                  returns (ptr),
  45       34+c 0006         dbuff  entry                  returns (ptr),
  46       35+c 0006         reboot entry,
  47       36+c 0006         rdcon  entry                  returns (char(1)),
  48       37+c 0006         wrcon  entry    (char(1)),
  49       38+c 0006         rdrdr  entry                  returns (char(1)),
  50       39+c 0006         wrpun  entry    (char(1)),
  51       40+c 0006         wrlst  entry    (char(1)),
  52       41+c 0006         coninp entry                  returns (char(1)),
  53       42+c 0006         conout entry    (char(1)),
  54       43+c 0006         rdstat entry                  returns (bit(1)),
  55       44+c 0006         getio  entry                  returns (bit(8)),
  56       45+c 0006         setio  entry    (bit(8)),
  57       46+c 0006         wrstr  entry    (ptr),
  58       47+c 0006         rdbuf  entry    (ptr),
  59       48+c 0006         break  entry                  returns (bit(1)),
```

```
  61      50+c 0006          reset  entry,
  62      51+c 0006          select entry    (fixed(7))  returns (bit(16)),
  63      52+c 0006          open   entry    (ptr)       returns (bit(16)),
  64      53+c 0006          close  entry    (ptr)       returns (bit(16)),
  65      54+c 0006          sear   entry    (ptr)       returns (bit(16)),
  66      55+c 0006          searn  entry                returns (bit(16)),
  67      56+c 0006          delete entry    (ptr)       returns (bit(16)),
  68      57+c 0006          rdseq  entry    (ptr)       returns (bit(16)),
  69      58+c 0006          wrseq  entry    (ptr)       returns (bit(16)),
  70      59+c 0006          make   entry    (ptr)       returns (bit(16)),
  71      60+c 0006          rename entry    (ptr)       returns (bit(16)),
  72      61+c 0006          logvec entry                returns (bit(16)),
  73      62+c 0006          curdsk entry                returns (fixed(7)),
  74      63+c 0006          setdma entry    (ptr),
  75      64+c 0006          allvec entry                returns (ptr),
  76      65+c 0006          wpdisk entry,
  77      66+c 0006          rovec  entry                returns (bit(16)),
  78      67+c 0006          filatt entry    (ptr),
  79      68+c 0006          getdpb entry                returns (ptr),
  80      69+c 0006          getusr entry                returns (fixed(7)),
  81      70+c 0006          setusr entry    (fixed(7)),
  82      71+c 0006          rdran  entry    (ptr)       returns (bit(16)),
  83      72+c 0006          wrran  entry    (ptr)       returns (bit(16)),
  84      73+c 0006          filsiz entry    (ptr),
  85      74+c 0006          setrec entry    (ptr),
  86      75+c 0006          resdrv entry    (bit(16))   returns (bit(16)),
  87      76+c 0006          wrranz entry    (ptr)       returns (bit(16));
  88      77+c 0006 /**** commented out for CCP/M-86 whf
  89      78+c 0006      dcl
  90      79+c 0006          testwr entry    (ptr)       returns (bit(16)),
  91      80+c 0006          lock   entry    (ptr)       returns (fixed(7)),
  92      81+c 0006          unlock entry    (ptr)       returns (fixed(7)),
  93      82+c 0006          multis entry    (fixed(7))  returns (fixed(7)),
  94      83+c 0006          ermode entry    (bit(1)),
  95      84+c 0006          freesp entry    (fixed(7))  returns (bit(16)),
  96      85+c 0006          chain  entry                returns (bit(16)),
  97      86+c 0006          flush  entry                returns (fixed(7)),
  98      87+c 0006          setlbl entry    (ptr)       returns (bit(16)),
  99      88+c 0006          getlbl entry    (fixed(7))  returns (bit(8)),
 100      89+c 0006          rdxfcb entry    (ptr)       returns (bit(16)),
 101      90+c 0006          wrxfcb entry    (ptr)       returns (bit(16)),
 102      91+c 0006          settod entry    (ptr),
 103      92+c 0006          gettod entry    (ptr),
 104      93+c 0006          dfpswd entry    (ptr),
 105      94+c 0006          sgscb  entry    (ptr) returns(bit(8));
 106      95 c 0006 ****/
 107      96 c 0006
 108      97 c 0006
 109      98+c 0006      declare
 110      99+c 0006          seldsk  entry   (fixed(7)) returns(ptr),
 111     100+c 0006          settrk  entry   (fixed(15)),
 112     101+c 0006          setsec  entry   (fixed(15)),
 113     102+c 0006          rdsec   entry   returns(fixed(7)),
 114     103+c 0006          wrsec   entry   (fixed(7)) returns(fixed(7)),
 115     104+c 0006          sectrn  entry   (fixed(15), ptr) returns(fixed(15)),
 116     105+c 0006          bstdma  entry   (ptr);
 117     106 c 0006
 118     107+c 0006      declare     /* CCPM special functions   whf 1/14/82 */
 119     108+c 0006          openvec   entry returns(fixed(15)),
 120     109+c 0006          syslock   entry returns(fixe(7)),
```

```
 121     110+c 0006          sysunlock entry,
 122     111+c 0006          conlock   entry returns(fixed(7)),
 123     112+c 0006          conunlock entry;
 124     113 c 0006
 125     114 c 0006          %replace
 126     115 c 0006             TRUE           by '1'b,
 127     116 c 0006             FALSE          by '0'b;
 128     117 c 0006
 129     118 c 0006                                      /* directory array 4K */
 130     119 c 0006          declare
 131     120 c 0006                  1 dir_fcb(0:127),
 132     121 c 0006                   3 user          bit(8),
 133     122 c 0006                   3 rest(31)      char(1),
 134     123 c 0006
 135     124 c 0006                  1 outbuf(0:127),
 136     125 c 0006                   2 user          fixed(7),
 137     126 c 0006                   2 rest(31)      char(1),
 138     127 c 0006
 139     128 c 0006                  1 buffer2(0:127),
 140     129 c 0006                   2 user          bit(8),
 141     130 c 0006                   2 rest(31)      bit(8),
 142     131 c 0006
 143     132 c 0006                  1 outb(0:127)   based(outptr),
 144     133 c 0006                   2 rest          char(32),
 145     134 c 0006
 146     135 c 0006                  1 outb2(0:127)  based(outptr),
 147     136 c 0006                   2 user          bit(8),
 148     137 c 0006                   2 rest(31)      char(1),
 149     138 c 0006
 150     139 c 0006                  1 outb3(0:127)  based(outptr),
 151     140 c 0006                   2 user          fixed(7),
 152     141 c 0006                   2 rest(31)      bit(8),
 153     142 c 0006
 154     143 c 0006                  1 outb4(0:127)  based(outptr),
 155     144 c 0006                   2 sfcbm         char(1),
 156     145 c 0006                   2 sfcb(3),
 157     146 c 0006                    3 stamps       char(8),
 158     147 c 0006                    3 mode         bit(8),
 159     148 c 0006                    3 rest         char(1),
 160     149 c 0006                   2 frest         char(1),
 161     150 c 0006
 162     151 c 0006                  1 infcb(0:127)  based(dirptr),
 163     152 c 0006                   2 rest          char(32),
 164     153 c 0006
 165     154 c 0006                  1 infcb2(0:127) based(dirptr),
 166     155 c 0006                   2 user          char(1),
 167     156 c 0006                   2 name          char(11),
 168     157 c 0006                   2 pmode         bit(8),
 169     158 c 0006                   2 junk1         char(11),
 170     159 c 0006                   2 stamp         char(8),
 171     160 c 0006
 172     161 c 0006                 1 clearbuf(0:127) based(clearptr),
 173     162 c 0006                   2 rest          char(32),
 174     163 c 0006
 175     164 c 0006                  zeroes(31)       bit(8) static init((31)'00000000'b)
 176     165 c 0006          ;
 177     166 c 0006
 178     167 c 0006                                      /* directory array mask */
 179     168 c 0006          declare
 180     169 c 0006             1 dirm(0:127)       based(dirptr),
```

```
181     170 c 0006              3 user          fixed(7),
182     171 c 0006              3 fname         char(8),
183     172 c 0006              3 ftype         char(3),
184     173 c 0006              3 fext          bin fixed(7),
185     174 c 0006              3 fs1           bit(8),
186     175 c 0006              3 fs2           bit(8),
187     176 c 0006              3 frc           fixed(7),
188     177 c 0006              3 diskpass(8)   char(1),
189     178 c 0006              3 rest          char(8);
190     179 c 0006
191     180 c 0006          declare                         /* disk parameter header mask */
192     181 c 0006                  dphp            ptr ext,
193     182 c 0006                  1 dph_mask      based(dphp),
194     183 c 0006                   2 xlt1         ptr,
195     184 c 0006                   2 space1(3)    bit(8),     /********************/
196     185 c 0006                   2 mediaf       bit(8),     /* whf 1/8/83      */
197     186 c 0006           2 space2(2)    bit(8),     /********************/
198     187 c 0006                   2 dpbptr       ptr,
199     188 c 0006                   2 csvptr       ptr,
200     189 c 0006                   2 alvptr       ptr,
201     190 c 0006                   2 dirbcb       ptr,
202     191 c 0006                   2 dtabcb       ptr,
203     192 c 0006                   2 hash         ptr,
204     193 c 0006 /***                  2 hbank        ptr,  ***/
205     194 c 0006
206     195 c 0006          xlt             ptr;    /* save the xlt ptr because of F10 buffer */
207     196 c 0006
208     197 c 0006          declare                 /* disk parameter block mask */
209     198 c 0006             dpbp        ptr external,
210     199 c 0006             1 dpb_mask  based(dpbp),
211     200 c 0006               2 spt     fixed(15),
212     201 c 0006               2 blkshft fixed(7),
213     202 c 0006               2 blkmsk  fixed(7),
214     203 c 0006               2 extmsk  fixed(7),
215     204 c 0006               2 dsksiz  fixed(15),
216     205 c 0006               2 dirmax  fixed(15),
217     206 c 0006               2 diralv  bit(16),
218     207 c 0006               2 checked fixed(15),
219     208 c 0006               2 offset  fixed(15),
220     209 c 0006               2 physhf  fixed(7),
221     210 c 0006               2 phymsk  fixed(7),
222     211 c 0006
223     212 c 0006      (        dspt    decimal(7,0),
224     213 c 0006                dblk    decimal(7,0) ) external;
225     214 c 0006
226     215 c 0006          declare (
227     216 c 0006             dir_blks(32)   bit(8),
228     217 c 0006             errorcode      bit(16)) external;
229     218 c 0006
230     219 c 0006          declare (
231     220 c 0006                  MAXSAVE         bin fixed(15),
232     221 c 0006                  enddcnt         bin fixed(15),
233     222 c 0006                  nxfcb           bin fixed(15),
234     223 c 0006                  notsaved        bin fixed(15),
235     224 c 0006                  xptr            pointer) external,
236     225 c 0006
237     226 c 0006                  1 XFCBs(1)      based(xptr),
238     227 c 0006                   2 user         bin fixed(7),
239     228 c 0006                   2 name         char(11),
240     229 c 0006                   2 pmode        bit(8),
```

```
 241     230 c 0006                2 stamp         char(8);
 242     231 c 0006
 243     232 c 0006
 244     233 c 0006 declare
 245     234 c 0006                INITMSG         char(54) static initial
 246     235 c 0006                   ('INITDIR WILL ACTIVATE TIME STAMPS FOR SPECIFIED DRIVE.'),
 247     236 c 0006                CONFIRM         char(60) varying static initial
 248     237 c 0006                   ('Do you want to re-format the directory on drive: '),
 249     238 c 0006
 250     239 c 0006                ASKCLEAR        char(44) static initial
 251     240 c 0006                   ('Do you want the existing time stamps cleared'),
 252     241 c 0006                RECOVER         char(50) varying static init
 253     242 c 0006                   ('Do you want to recover time/date directory space'),
 254     243 c 0006                YN              char(10) static initial('  (Y/N)?  '),
 255     244 c 0006                YES             char(1) static initial('Y'),
 256     245 c 0006                lyes            char(1) static initial('y'),
 257     246 c 0006                yesno           char(1),
 258     247 c 0006
 259     248 c 0006                UPPERCASE       char(26) static initial
 260     249 c 0006                                ('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
 261     250 c 0006                LOWERCASE       char(26) static initial
 262     251 c 0006                                ('abcdefghijklmnopqrstuvwxyz'),
 263     252 c 0006
 264     253 c 0006                pass1           char(20) static initial
 265     254 c 0006                                ('End of PASS 1.'),
 266     255 c 0006                ERRORM          char(7) static initial
 267     256 c 0006                ('ERROR: '),
 268     257 c 0006                TERM            char(30) static initial
 269     258 c 0006                ('INITDIR TERMINATED.'),
 270     259 c 0006                errvers         char(50) static initial
 271     260 c 0006                                ('Requires Concurrent CP/M-86 2.0'),
 272     261 c 0006                errnotnew       char(31) static initial
 273     262 c 0006                                ('Directory already re-formatted.'),
 274     263 c 0006                errtoobig       char(30) static initial
 275     264 c 0006                                ('Not enough room in directory.'),
 276     265 c 0006                errpass         char(15) static initial
 277     266 c 0006                ('Wrong password.'),
 278     267 c 0006                errSTRIP        char(30) varying static initial
 279     268 c 0006                                ('No time stamps present.'),
 280     269 c 0006                errMEM          char(30) varying static initial
 281     270 c 0006                                ('Not enough available memory.'),
 282     271 c 0006                errRO           char(20) varying static initial
 283     272 c 0006                                ('Disk is READ ONLY.'),
 284     273 c 0006                errWHAT         char(30) varying static initial
 285     274 c 0006                                ('Cannot find last XFCB.'),
 286     275 c 0006 /***           errRSX          char(60) varying static initial
 287     276 c 0006                ('Cannot re-format the directory with RSXs in memory.'), ***/
 288     277 c 0006                errunrec        char(19) static initial
 289     278 c 0006                ('Unrecognized drive.'),
 290     279 c 0006          errDISKACT  char(40) varying static initial
 291     280 c 0006                ('Some other process has an open file.'),
 292     281 c 0006          errCONSOLE  char(40) varying static initial
 293     282 c 0006                ('INITDIR must be run in foreground only.'),
 294     283 c 0006          errXREAD        char(22) varying static initial
 295     284 c 0006                ('Fatal XIOS read error.'),
 296     285 c 0006          errXWRITE       char(23) varying static initial
 297     286 c 0006                ('Fatal XIOS write error.'),
 298     287 c 0006                errBIOS         char(20) static initial
 299     288 c 0006                ('Cannot select drive.');
 300     289 c 0006
```

```
  291     c 0006          declare (
  292     c 0006                  outptr          pointer,
  293     c 0006                  bufptr1         pointer,
  294     c 0006                  bufptr2         pointer,
  295     c 0006                  dirptr          pointer,
  296     c 0006                  drivptr         pointer,
  297     c 0006                  clearptr        pointer,
  298     c 0006
  299     c 0006                  nempty          bin fixed(15),
  300     c 0006                  (nfcbs,nfcbs1)  bin fixed(15),
  301     c 0006                  lastsfcb        bin fixed(15),
  302     c 0006                  lastdcnt        bin fixed(15),
  303     c 0006                  (lasti,lastx)   bin fixed(15),
  304     c 0006                  lastsect        bin fixed(15),
  305     c 0006                  cleardcnt       bin fixed(15),
  306     c 0006                  (gsec,gtrk)     bin fixed(15),
  307     c 0006                  (dcnt,sect)     bin fixed(15),
  308     c 0006                  outdcnt         bin fixed(15),
  309     c 0006                  newdcnt         bin fixed(15),
  310     c 0006                  outidx          bin fixed(7),
  311     c 0006                  curdisk         bin fixed(7),
  312     c 0006                  newlasti        bin fixed(7),
  313     c 0006                  (sfcbidx,sfcboffs)  bin fixed(15),
  314     c 0006                  usernum         fixed(7),
  315     c 0006                  SFCBmark        fixed(7) static initial(33),
  316     c 0006                  Dlabel          bin fixed(7) static initial (32)
  317     c 0006      ) external,
  318     c 0006
  319     c 0006                  Redo            bit(1),
  320     c 0006                  bad             bit(1),
  321     c 0006                  writeflag       bit(1),
  322     c 0006                  CLEARSECT       bit(1),
  323     c 0006                  CLEARSFCB       bit(1),
  324     c 0006                  labdone         bit(1) static initial(false),
  325     c 0006                  cversion        bit(16),
  326     c 0006                  READonly        bit(16),
  327     c 0006
  328     c 0006                  ptreos          pointer,
  329     c 0006                  EOS             bit(8) static initial('00'b4),
  330     c 0006                  CEOS            char(1) based (ptreos),
  331     c 0006
  332     c 0006                  fcb(32)         char(1),
  333     c 0006                  fcb0(50)        char(1) based (drivptr),
  334     c 0006                  dr0             fixed(7) based(drivptr),
  335     c 0006                  disks           char(16) static initial
  336     c 0006                                  ('ABCDEFGHIJKLMNOP'),
  337     c 0006                  drive           bin fixed(7),
  338     c 0006                  cdrive          char(1);
  339     c 0006
  340     c 0006  declare
  341     c 0006                  1 SCB,
  342     c 0006                   2 soffs        fixed(7),
  343     c 0006                   2 seter        fixed(7),
  344     c 0006                   2 value        char(2);
  345     c 0006
  346     c 0006  /**** commented out whf CCP/M-86
  347     c 0006      dcl
  348     c 0006                  ccppage         bit(3);
  349     c 0006  ****/
  350     c 0006
```

```
 361     350 c 0006 /*****************************************************************************
 362     351 c 0006
 363     352 c 0006
 364     353 c 0006                          ***  MAIN PROGRAM  ***
 365     354 c 0006
 366     355 c 0006
 367     356 c 0006 *****************************************************************************/
 368     357 c 0006
 369     358 c 0006         declare i       bin fixed(7);
 370     359 c 0006
 371     360 c 0006         cversion = vers();
 372     361 c 000D         if substr(cversion,9,8) ^= "31"b4    |
 373     362 c 0043         ((substr(cversion,1,8) & "11111101"b1) ^= "14"b4)
 374     363 c 0043             then call errprint((errvers));
 375     364 c 0056
 376     365 c 0056     if openvec() ^= 0 then call errprint(errDISKACT);  /*** 1/83 whf ***/
 377     366 c 006F
 378     367 c 006F         soffs = 23;
 379     368 c 0074         seter = 0;
 380     369 c 0079 /***         ccppage = sgscb(addr(SCB));           /* if RSX present then stop */
 381     370 c 0079 /***         if substr(ccppage,7,1) = "1"b then call errprint(errRSX);   */
 382     371 c 0079
 383     372 c 0079         drivptr = dfcb0();                        /* get drive */
 384     373 c 0080         drive = dr0;
 385     374 c 0089         if dr0 > 16  then drive = 0;
 386     375 c 0097
 387     376 c 0097         do while(drive = 0);                      /* none recognized */
 388     377 c 009E                 call wrongdisk(1,drive);
 389     378 c 00A4                 call getdisk(1,drive);
 390     379 c 00AC         end;
 391     380 c 00AC
 392     381 c 00AC         cdrive = substr(disks,drive,1);
 393     382 c 00C6
 394     383 c 00C6         curdisk = curdsk();                       /* restore BIOS to this */
 395     384 c 00CC
 396     385 c 00CC         put edit(INITMSG,confirm,cdrive,YN)(skip(2),a,skip,a,a,a);
 397     386 c 0108         get list(yesno);
 398     387 c 0121         if yesno ^= YES & yesno ^= lyes then call reboot;
 399     388 c 0154
 400     389 c 0154         READonly = rovec();                       /* is the drive RO ? */
 401     390 c 0158         if substr(READonly,(17-drive),1) = "1"b then
 402     391 e 0180            call errprint(errRO);
 403     392 e 0191
 404     393 e 0191         call dselect(drive);
 405     394 e 0197         nfcbs = ((phymsk + 1)*4) - 1;             /* # fcbs/physical rcd - 1 */
 406     395 e 01A9         nfcbs1 = nfcbs + 1;
 407     396 e 01AD
 408     397 e 01AD     /* everything kosher, lock up system */
 409     398 e 01AD     if syslock() ^= 0 then call errprint(errDISKACT);  /*** 1/83 whf ***/
 410     399 e 01C5
 411     400 e 01C5         dirptr = addr(dir_fcb(0));
 412     401 e 01CB         dcnt = 0;
 413     402 e 01D1         call read_sector(dcnt,dirptr);
 414     403 e 01D7
 415     404 e 01D7         call allxfcb;                          /* allocate XFCB data space */
 416     405 e 01DA     if dirm(3).user = SFCBmark then call query;   /* recover SFCB space? */
 417     406 e 01EA         call countdir;           /* count number of directory entries */
 418     407 e 01ED
 419     408 e 01ED     if conlock() ^= 0 then call errprint(errCONSOLE);  /*** 1/83 whf ***/
 420     409 e 0205
```

```
421     410 e 0205          call init;
422     411 e 0208          call restore;
423     412 e 0208
424     413 e 0208 /*************************************************************************/
425     414 e 0208
426     415 e 0208
427     416 e 0208 wrongdisk: procedure(i,drive) external;
428     417 e 0208          declare (i,j,drive)     bin fixed(7);
429     418 e 0218
430     419 e 0218          put list(ERROR#,errunrec);
431     420 e 023A
432     421 e 023A /**       put skip list('DRIVE: ');
433     422 e 023A          j = i;
434     423 e 023A          ptreos = addr(EOS);
435     424 e 023A          do while(fcb0(j) ^= ' ' & fcb0(j) ^= EOS);
436     425 e 023A                  put edit(fcb0(j))(a);
437     426 e 023A                  j = j + 1;
438     427 e 023A          end;
439     428 e 023A **/
440     429 e 023A          put skip;
441     430 c 024C
442     431 c 024C end wrongdisk;
443     432 e 024C
444     433 e 024C getdisk: procedure(i,drive) external;
445     434 e 024C          declare (i,drive)       bin fixed(7);
446     435 e 025A
447     436 e 025A          put skip list('Enter Drive: ');
448     437 e 0276          get list(fcb0(i));
449     438 e 02A1          fcb0(i) = translate(fcb0(i),UPPERCASE,LOWERCASE);
450     439 e 02EA          fcb0(i+1) = ':';
451     440 e 0307
452     441 e 0307          drive = index(disks,fcb0(i));
453     442 c 0339
454     443 c 0339 end getdisk;
455     444 e 0339
456     445 e 0339
457     446 e 0339 /*************************************************************************/
458     447 e 0339
459     448 e 0339
460     449 e 0339 init: procedure external;
461     450 e 0339
462     451 e 0339          declare
463     452 e 033C                  (i,j,k,l)       bin fixed(15);
464     453 e 033C
465     454 e 033C          lastx = nxfcb;
466     455 e 0342          sect = sect - 1;
467     456 e 0346          dcnt = dcnt - 1;                        /* reset to good dcnt */
468     457 e 034A
469     458 e 034A          if Redo then do;
470     459 e 0353                  newdcnt = lastdcnt;
471     460 e 0359                  newlasti = lasti;
472     461 e 0362          end;
473     462 e 0362          else do;
474     463 e 0362                  lastsfcb = lastdcnt/3 + 1;
475     464 e 036F                  newdcnt = lastdcnt + lastsfcb + (2 - mod(lastdcnt,3));
476     465 e 0390                  if newdcnt > dirmax then do;
477     466 e 03A1                          lastdcnt = dirmax - nempty;
478     467 e 03B0                          lastsfcb = lastdcnt/3 + 1;
479     468 e 03BA                          newdcnt = lastdcnt + lastsfcb + (2 - mod(lastdcnt,3));
480     469 e 03D8
```

```
 481     470 e 03D8                        if newdcnt > dirmax then
 482     471 e 03E9                           call errprint(errtoobig);
 483     472 e 03FC
 484     473 e 03FC                        call collapse;           /* remove all empties by
 485     474 e 03FF                                                    collapsing dir from top */
 486     475 e 03FF                        lastsfcb = lastdcnt/3 + 1;
 487     476 e 040C                        newdcnt = lastdcnt + lastsfcb + (2 - mod(lastdcnt,3));
 488     477 e 042D                        if newdcnt > dirmax then
 489     478 e 043B                           call errprint(errtoobig);
 490     479 e 044E                    end;
 491     480 e 044E                    newlasti = mod(newdcnt,nfcbs1) - 3 + mod(lastdcnt,3);
 492     481 e 0472             end;
 493     482 e 0472
 494     483 e 0472             outptr = addr(buffer2(0));              /* want to clear last read
 495     484 e 0478                                                        sector...buffer2 only used
 496     485 e 0478                                                        in collapse so it is free */
 497     486 e 0478             call clearout;
 498     487 e 047B             clearptr = outptr;
 499     488 e 0481             outptr = addr(outbuf(0));
 500     489 e 0487             call clearout;                          /* zero output buffer */
 501     490 e 048A
 502     491 e 048A
 503     492 e 048A /*****************************************************************************/
 504     493 e 048A
 505     494 e 048A
 506     495 e 048A             do while(lastsect < sect );             /* clear from end of dir */
 507     496 e 0493                    call write_sector(dcnt,outptr);
 508     497 e 0499                    dcnt = dcnt - nfcbs1;
 509     498 e 04A0                    sect = sect - 1;
 510     499 e 04A6             end;
 511     500 e 04A6
 512     501 e 04A6             if (nempty - 1) ~= dirmax then do;      /* if there are files on dir */
 513     502 e 04B4
 514     503 e 04B4                                                     /* bottom of directory is
 515     504 e 04B4                                                        now all E5 and 21...
 516     505 e 04B4                                                        it is positioned to the
 517     506 e 04B4                                                        last good sector of the old
 518     507 e 04B4                                                        directory.  */
 519     508 e 04B4                dcnt = lastdcnt;
 520     509 e 04BA                enddcnt = newdcnt;
 521     510 e 04C0                call read_sector(dcnt,dirptr);       /* read last good sector */
 522     511 e 04C6
 523     512 e 04C6                outidx = newlasti;                   /* index into out buffer */
 524     513 e 04CC                call buildnew(lasti);                /* fill in outbuff from the
 525     514 e 04D2                                                        bottom up...need this call
 526     515 e 04D2                                                        because lasti may be in
 527     516 e 04D2                                                        middle of read buffer */
 528     517 e 04D2                do while(dcnt >= 0);
 529     518 e 04D9                                                     /* as soon as we are finished
 530     519 e 04D9                                                        with reading old sector,
 531     520 e 04D9                                                        then go clear it.  This
 532     521 e 04D9                                                        should limit possibility
 533     522 e 04D9                                                        that duplicate FCB's occur.
 534     523 e 04D9                                                     */
 535     524 e 04D9                     call read_sector(dcnt,dirptr);
 536     525 e 04DF                     call buildnew(nfcbs);
 537     526 e 04E9                end;
 538     527 e 04E9
 539     528 e 04E9             end;                                    /* virgin dir */
 540     529 e 04E9
```

```
 541     530 e 04E9         else call write_sector(0,outptr);       /* write last sector */
 542     531 e 04EF
 543     532 e 04EF         do while(notsaved > 0);
 544     533 e 04F6                 call moreXFCB;
 545     534 c 04FC         end;
 546     535 c 04FC
 547     536 c 04FC end init;
 548     537 e 04FC
 549     538 e 04FC /*****************************************************************************/
 550     539 e 04FC
 551     540 e 04FC
 552     541 e 04FC strip: procedure external;
 553     542 e 04FC
 554     543 e 04FC                                 /* remove all SFCB from directory by jamming
 555     544 e 04FC                                    E5 into user field.  Also turn off time/date
 556     545 e 04FC                                    stamping in DIR LABEL.  */
 557     546 e 04FC
 558     547 e 04FC         declare (i,j)           bin fixed(7),
 559     548 e 04FF                 1 direct(0:127) based(dirptr),
 560     549 e 04FF                  2 junk1        char(12),
 561     550 e 04FF                  2 ext          bit(8),
 562     551 e 04FF                  2 rest         char(19),
 563     552 e 04FF
 564     553 e 04FF                 olddcnt         bin fixed(15);
 565     554 e 04FF
 566     555 e 04FF
 567     556 e 04FF         dcnt = 0;
 568     557 e 0505
 569     558 e 0505         do while(dcnt <= dirmax);
 570     559 e 0516
 571     560 e 0516                 call read_sector(dcnt,dirptr);
 572     561 e 051C
 573     562 e 051C                 olddcnt = dcnt;
 574     563 e 0522                 do i = 0 to nfcbs while(dcnt <= dirmax);
 575     564 e 0555
 576     565 e 0555                         if ^labdone then
 577     566 e 0560                             if dirm(i).user = Dlabel then do;
 578     567 e 0575                                 call getpass(i);
 579     568 e 0578                                 direct(i).ext = direct(i).ext & '10000001'b;
 580     569 e 058E                                 labdone = true;
 581     570 e 0593                             end;
 582     571 e 0593
 583     572 e 0593                         if dirm(i).user = SFCBmark then
 584     573 e 05A8                             dir_fcb(i).user = 'E5'b4;
 585     574 e 05B7
 586     575 e 05B7                         dcnt = dcnt + 1;
 587     576 e 05C2                 end;
 588     577 e 05C2
 589     578 e 05C2                 call write_sector(olddcnt,dirptr);
 590     579 c 05CC         end;
 591     580 c 05CC
 592     581 c 05CC end strip;
 593     582 e 05CC
 594     583 e 05CC
 595     584 e 05CC /*****************************************************************************/
 596     585 e 05CC
 597     586 e 05CC
 598     587 e 05CC
 599     588 e 05CC countdir: procedure external;
 600     589 e 05CC         declare i       bin fixed(7);
```

```
601   590 e 05CF
602   591 e 05CF
603   592 e 05CF
604   593 e 05CF
605   594 e 05CF
606   595 e 05CF
607   596 e 05CF
608   597 e 05CF
609   598 e 05CF
610   599 e 05CF
611   600 e 05CF
612   601 e 05CF
613   602 e 05CF
614   603 e 05CF
615   604 e 05CF
616   605 e 05CF      Redo = false;
617   606 e 05D4      nempty = 0;
618   607 e 05DA      sect = 0;
619   608 e 05E0      nxfcb = 0;
620   609 e 05E6      notsaved = 0;
621   610 e 05EC      bad = true;
622   611 e 05F1
623   612 e 05F1
624   613 e 05F1
625   614 e 05F1
626   615 e 05F1
627   616 e 05F1
628   617 e 05F1
629   618 e 05F1
630   619 e 05F1
631   620 e 05F1
632   621 e 05F1      if dirm(3).user = SFCBmark then bad = false;
633   622 e 0603
634   623 e 0603      do while(dcnt <= dirmax);
635   624 e 0614              do i = 0 to nfcbs while(dcnt <= dirmax);
636   625 e 0647                      if dir_fcb(i).user ^= 'E5'b4 then do;
637   626 e 0663                              usernum = dirm(i).user;
638   627 e 0676
639   628 e 0676              /* assume sfcb's were caught before this */
640   629 e 0676                              if usernum > 15 & usernum < 32 then
641   630 e 0692                                 call getXFCB(i);
642   631 e 069A
643   632 e 069A                                      /* if LABEL then check for password...
644   633 e 069A                                         may terminate in getpass */
645   634 e 069A
646   635 e 069A                              else if usernum = 0label then call getpass(i);
647   636 e 06A9
648   637 e 06A9                              if (usernum < 33) | (^bad & usernum = 33) then
649   638 e 06CE                                      do;
650   639 e 06CE
651   640 e 06CE                                      lasti = i;
652   641 e 06D5                                      lastsect = sect;
653   642 e 06DB                                      lastdcnt = dcnt;
654   643 e 06E3                              end;                    /* bad...*/
655   644 e 06E3                              else if usernum = 33 then nempty = nempty + 1;
656   645 e 06F0
657   646 e 06F0                      end;                            /* E5 ... */
658   647 e 06F0                      else nempty = nempty + 1;
659   648 e 05F4                      dcnt = dcnt + 1;
660   649 e 06FF              end;
```

```
/* there are 5 valid sets of codes in
   the user field:

        E5      - empty
        0-15    - user numbers
        32      - Directory label
        33      - SFCB marker
        16-31   - XFCB marker

   This routine counts the # of used
   directory slots ignoring E5.
   NOTE: if SFCB present then last
         slot = SFCB */
```

```
/* If dir is already time stamped then
   SFCBs should appear in every sector,
   notably the first sector. Thus,
   test first sector.  If first sector
   has SFCB then all do.  If none in
   first & they appear later then
   INITDIR was probably interrupted.
   In that case, zap the found SFCB's
   and treat dir as virgin.  */
```

```
 661     650 e 06FF
 662     651 e 06FF                 sect = sect + 1;
 663     652 e 0703                 call read_sector(dcnt,dirptr);
 664     653 e 070C         end;
 665     654 e 070C
 666     655 e 070C         if ^Redo then lastsfcb = lastdcnt/3 + 1;
 667     656 c 0725
 668     657 c 0725 end countdir;
 669     658 e 0725
 670     659 e 0725 getXFCB: procedure(i) external;
 671     660 e 0725         declare i       bin fixed(7);
 672     661 e 072D
 673     662 e 072D          if nxfcb <= MAXSAVE then do;
 674     663 e 0739                 nxfcb = nxfcb + 1;
 675     664 e 073D                 XFCBs(nxfcb).user = usernum - 16;
 676     665 e 0755                 XFCBs(nxfcb).name = infcb2(i).name;
 677     666 e 0786                 XFCBs(nxfcb).pmode = infcb2(i).pmode;
 678     667 e 07AD                 XFCBs(nxfcb).stamp = infcb2(i).stamp;
 679     668 e 07E1          end;
 680     669 e 07E1          else notsaved = notsaved + 1;
 681     670 c 07E6
 682     671 c 07E6 end getXFCB;
 683     672 e 07E6
 684     673 e 07E6
 685     674 e 07E6 allxfcb: procedure external;
 686     675 e 07E6
 687     676 e 07E6                                 /* allocates largest available block of space
 688     677 e 07E6                                    to be used in storing XFCB info.
 689     678 e 07E6                                         maxwds & allwds use word units */
 690     679 e 07E6
 691     680 e 07E6         declare maxwds          entry returns(fixed(15)),
 692     681 e 07E8                 allwds          entry(fixed(15)) returns(pointer),
 693     682 e 07E8                 size            bin fixed(15);
 694     683 e 07E8
 695     684 e 07E8         size = maxwds();                /* get largest block in free space */
 696     685 e 07EF         xptr = allwds(size);            /* reserve it */
 697     686 e 07F9         MAXSAVE = 2*(size/21);          /* # XFCBs that can be saved */
 698     687 e 0807         if MAXSAVE <= 10 then call errprint(errMEM);
 699     688 c 0820
 700     689 c 0820 end allxfcb;
 701     690 e 0820
 702     691 e 0820
 703     692 e 0820 query: procedure external;
 704     693 e 0820
 705     694 e 0820         if bad then return;
 706     695 e 082C
 707     696 e 082C         put skip(2) list(errnotnew);
 708     697 e 0848
 709     698 e 0848                                                 /* check to see if user wants
 710     699 e 0848                                                    to strip SFCB's */
 711     700 e 0848         if ^asker(RECOVER) then do;
 712     701 e 0861                 Redo = true;
 713     702 e 0866                 CLEARSFCB = false;
 714     703 e 086B                 if asker(ASKCLEAR) then do;
 715     704 e 0882                         CLEARSFCB = true;
 716     705 e 0887                         return;
 717     706 e 088A                 end;
 718     707 e 088A         end;
 719     708 e 088A         else call strip;                       /* this will end down here
 720     709 e 088D                                                    after stripping */
```

```
721     710 e 088D
722     711 e 088D          call restore;                       /* dir is already formattted &
723     712 c 0891                                                 user does not want to clear
724     713 c 0891                                                 old SFCB's....just stop */
725     714 c 0891
726     715 c 0891  end query;
727     716 e 0891
728     717 e 0891  buildnew: procedure(endidx) external;
729     718 e 0891          declare (i,j,k,endidx)          bin fixed(15);
730     719 e 0899
731     720 e 0899  declare 1 ot(0:127)     based(outptr),
732     721 e 0899           2 user         fixed(7),
733     722 e 0899           2 fname        char(8),
734     723 e 0899           2 ftype        char(3),
735     724 e 0899           2 rest         char(20);
736     725 e 0899
737     726 e 0899                                      /* build output buffer from
738     727 e 0899                                         input(end) to input(0).
739     728 e 0899                                         k => refers to input */
740     729 e 0899          k = endidx;
741     730 e 08A2          do while(k >= 0);
742     731 e 08AC                  usernum = dirm(k).user;
743     732 e 08BE
744     733 e 08BE                  outb(outidx).rest = infcb(k).rest;
745     734 e 08E8
746     735 e 08E8                  if usernum = SFCBmark then do;
747     736 e 08F1                          if bad then outb2(outidx).user = 'E5'b4;
748     737 e 090D                          else if CLEARSFCB then outb3(outidx).rest = zeroes;
749     738 e 0935                  end;
750     739 e 0935
751     740 e 0935                  if usernum < 16 then do;
752     741 e 093C                     if nxfcb > 0 then            /* if fcb is ex=0 and XFCB
753     742 e 0943                                                     exists then check for
754     743 e 0943                                                     possible SFCB update */
755     744 e 0943                          call putXFCB(k);
756     745 e 094F                  end;
757     746 e 094F
758     747 e 094F                  if ^Redo & mod(outidx,4) = 0 then outidx = outidx - 2;
759     748 e 0980                  else outidx = outidx - 1;
760     749 e 0984
761     750 e 0984                  k = k - 1;
762     751 e 0988                  dcnt = dcnt - 1;
763     752 e 098C
764     753 e 098C                  if outidx < 0 then do;
765     754 e 0993                          if dcnt > 14 then
766     755 e 099A                             if mod(dcnt + 1,nfcbs1) = 0 then
767     756 e 09AC                                call write_sector(dcnt + 1,clearptr);
768     757 e 09B9                          call write_sector(newdcnt,outptr);
769     758 e 09BF                          newdcnt = newdcnt - nfcbs1;
770     759 e 09C6                          outidx = nfcbs - 1;
771     760 e 09CD                          if Redo then outidx = outidx + 1;
772     761 c 09DE                  end;
773     762 c 09DE          end;
774     763 c 09DE
775     764 c 09DE  end buildnew;
776     765 e 09DE
777     766 e 09DE
778     767 e 09DE  /*************************************************************************/
779     768 e 09DE
780     769 e 09DE
```

```
 781     770 e 09DE compare: procedure(k) returns(fixed(7)) external;
 782     771 e 09DE
 783     772 e 09DE         declare (i,j,k)          bin fixed(7),
 784     773 e 09E6                 1 direc(0:127)   based(dirptr),
 785     774 e 09E6                  2 user          fixed(7),
 786     775 e 09E6                  2 name(11)      char(1),
 787     776 e 09E6                  2 rest          char(20),
 788     777 e 09E6
 789     778 e 09E6                 1 XFCB2(1)       based(xptr),
 790     779 e 09E6                  2 user          char(1),
 791     780 e 09E6                  2 name(11)      char(1),
 792     781 e 09E6                  2 rest          char(9);
 793     782 e 09E6
 794     783 e 09E6                                          /* compare fcb with XFCB list;
 795     784 e 09E6                                             return position in list if
 796     785 e 09E6                                             found, 0 otherwise.
 797     786 e 09E6                                             Nullify usernum field in
 798     787 e 09E6                                             XFCB list (=99) if found.
 799     788 e 09E6                                             Decrement #xfcb as well.*/
 800     789 e 09E6         do i = 1 to nxfcb;
 801     790 e 09FD                 if XFCBs(i).user ^= 99 then do;
 802     791 e 0A18                         if XFCBs(i).user = direc(k).user then do;
 803     792 e 0A45
 804     793 e 0A45                                 do j = 1 to 11;
 805     794 e 0A51                                    if direc(k).name(j) ^= XFCB2(i).name(j)
 806     795 e 0A94                                       then go to outx;
 807     796 e 0A9C                                 end;
 808     797 e 0A9C
 809     798 e 0A9C                                 /* found a match */
 810     799 e 0A9C                                 XFCBs(i).user = 99;
 811     800 e 0AB2                                 nxfcb = nxfcb - 1;
 812     801 e 0AB6                                 return(1);
 813     802 e 0AC1
 814     803 e 0AC1 outx:                   end;
 815     804 e 0AC1                 end;
 816     805 e 0AC1         end;
 817     806 e 0AC1
 818     807 e 0AC1         return(0);
 819     808 c 0AC4
 820     809 c 0AC4 end compare;
 821     810 e 0AC4
 822     811 e 0AC4 moreXFCB: procedure external;
 823     812 e 0AC4                                 /* we could not store all the xfcb's in memory
 824     813 e 0AC4                                    available, so now must make another pass &
 825     814 e 0AC4                                    store as many XFCB as possible.
 826     815 e 0AC4                                    'notsaved' > 0 ==> we may have to
 827     816 e 0AC4                                                       do this again.  */
 828     817 e 0AC4         declare (i,k)   bin fixed(7);
 829     818 e 0AC7
 830     819 e 0AC7         dcnt = enddcnt;                 /* go to end of directory */
 831     820 e 0ACD         if ^findXFCB(k) then            /* work backwards trying to find
 832     821 e 0ADB                                            last known XFCB...if not found
 833     822 e 0ADB                                            then something very strange has
 834     823 e 0ADB                                            happened; */
 835     824 e 0ADB             call errprint(errWHAT);
 836     825 e 0AEC
 837     826 e 0AEC         notsaved = 0;                   /* now in last sector where last XFCB
 838     827 e 0AF2                                            occurs...look for other XFCB that
 839     828 e 0AF2                                            we know is there.  */
 840     829 e 0AF2         nxfcb = 0;
```

```
 841     830 e 0AF8
 842     831 e 0AF8          dcnt = dcnt + 1;
 843     832 e 0AFC          lastdcnt = dcnt;                /* save position of last XFCB + 1 */
 844     833 e 0B02          lasti = k + 1;                  /* index in sector */
 845     834 e 0B0A          do while(dcnt <= enddcnt);
 846     835 e 0B16                  do i = k+1 to nfcbs while(dcnt <= enddcnt);
 847     836 e 0B45                          usernum = dirm(i).user;
 848     837 e 0B58                          if usernum > 15 & usernum < 32 then call getXFCB(i);
 849     838 e 0B7A                          dcnt = dcnt + 1;
 850     839 e 0B84                  end;
 851     840 e 0B84                  k = 0;
 852     841 e 0B89                  call read_sector(dcnt,dirptr);
 853     842 e 0B92          end;
 854     843 e 0B92
 855     844 e 0B92          dcnt = 0;                               /* go to start of dir */
 856     845 e 0B98          do while(dcnt <= enddcnt);
 857     846 e 0BA4                  call read_sector(dcnt,dirptr);
 858     847 e 0BAA                  outdcnt = dcnt;
 859     848 e 0BB0                  writeflag = false;              /* putXFCB sets when it finds a
 860     849 e 0BB5                                                     match */
 861     850 e 0BB5
 862     851 e 0BB5                  do k = 0 to nfcbs while(dcnt <= enddcnt);
 863     852 e 0BE1                          outidx = k;
 864     853 e 0BE7                          if dirm(k).user < 16 then call putXFCB(k);
 865     854 e 0C00                          dcnt = dcnt + 1;
 866     855 e 0C0A                  end;
 867     856 e 0C0A                  if writeflag then call write_sector(outdcnt,dirptr);
 868     857 c 0C1D          end;
 869     858 c 0C1D
 870     859 c 0C1D end moreXFCB;
 871     860 e 0C1D
 872     861 e 0C1D findXFCB: procedure(idx) returns(bit(1)) external;
 873     862 e 0C1D
 874     863 e 0C1D                          /* find the last known XFCB...starts from the
 875     864 e 0C1D                             last written sector in the dir and goes
 876     865 e 0C1D                             backwards...hopefully that's faster */
 877     866 e 0C1D          declare idx     fixed(7);
 878     867 e 0C25
 879     868 e 0C25          do while(dcnt > 0);
 880     869 e 0C2F                  call read_sector(dcnt,dirptr);
 881     870 e 0C35                  do idx = 0 to nfcbs while(dcnt > 0);
 882     871 e 0C65                          usernum = dirm(idx).user;
 883     872 e 0C7B                          if usernum > 15 & usernum < 32 then
 884     873 e 0C97                                  if XFCBs(lastx).name = infcb2(idx).name then
 885     874 e 0CCC                                          return(true);
 886     875 e 0CCF                          dcnt = dcnt - 1;
 887     876 e 0CDF                  end;
 888     877 e 0CDF          end;
 889     878 e 0CDF
 890     879 e 0CDF          return(false);          /* big trouble...*/
 891     880 c 0CE2
 892     881 c 0CE2 end findXFCB;
 893     882 e 0CE2
 894     883 e 0CE2
 895     884 e 0CE2 putXFCB: procedure(k) external;
 896     885 e 0CE2                          /* if this is extent 0 fold and names match
 897     886 e 0CE2                             then update SFCB from XFCB */
 898     887 e 0CE2          declare (k,j)   fixed(7);
 899     888 e 0CEA
 900     889 e 0CEA                          if dirm(k).fext <= dph_mask.extmsk then do;
```

```
 901     890 e 0D0C                          j = compare(k);
 902     891 e 0D1B                          if j ~= 0 then do;
 903     892 e 0D25
 904     893 e 0D25                                      /* fcb matches XFCB...
 905     894 e 0D25                                         update the SFCB */
 906     895 e 0D25                            sfcboffs = mod(outidx+1,4);
 907     896 e 0D36                            sfcbidx = outidx + (4 - sfcboffs);
 908     897 e 0D45                            outb4(sfcbidx).sfcb(sfcboffs).stamps =
 909     898 e 0D7F                                                    XFCBs(j).stamp;
 910     899 e 0D7F                            outb4(sfcbidx).sfcb(sfcboffs).mode =
 911     900 e 0DAE                                                    XFCBs(j).pmode;
 912     901 e 0DAE                            writeflag = true;
 913     902 c 0DB4                          end;
 914     903 c 0DB4                        end;                      /* extent 0 ? */
 915     904 c 0DB4
 916     905 c 0DB4 end putXFCB;
 917     906 e 0DB4
 918     907 e 0DB4
 919     908 e 0DB4 errprint: procedure(msg) external;
 920     909 e 0DB4          declare
 921     910 e 0DBB                  msg             char(60) varying;
 922     911 e 0DBB
 923     912 e 0DBB          put edit(ERRORM,msg,TERM)(skip(2),a,a,skip,a);
 924     913 e 0DEF          put skip(2);
 925     914 e 0E00
 926     915 e 0E00          call restore;
 927     916 c 0E04
 928     917 c 0E04 end errprint;
 929     918 e 0E04
 930     919 e 0E04
 931     920 e 0E04 asker: procedure(msg) returns(bit(1)) external;
 932     921 e 0E04
 933     922 e 0E04          declare msg             char(60) varying;
 934     923 e 0E0C
 935     924 e 0E0C          put skip list(msg,YN);
 936     925 e 0E34          get list(yesno);
 937     926 e 0E4D
 938     927 e 0E4D          if yesno ~= YES & yesno ~= lyes then return(false);
 939     928 e 0E80
 940     929 e 0E80          return(true);
 941     930 c 0E83
 942     931 c 0E83 end asker;
 943     932 e 0E83
 944     933 e 0E83
 945     934 e 0E83 clearout: procedure external;
 946     935 e 0E83          declare
 947     936 e 0E86                  (i,j)   bin fixed(7);
 948     937 e 0E86
 949     938 e 0E86          do i = 0 to nfcbs;
 950     939 e 0E9A                  if mod(i+1,4) ~= 0 then outb2(i).user = 'E5'b4;
 951     940 e 0EBF                  else outb3(i).user = SFCBmark;
 952     941 e 0ED2
 953     942 e 0ED2                  do j = 1 to 31;
 954     943 e 0EDE                      outb3(i).rest(j) = '00000000'b;
 955     944 c 0F02                  end;
 956     945 c 0F02          end;
 957     946 c 0F02
 958     947 c 0F02 end clearout;
 959     948 e 0F02
 960     949 e 0F02 getpass: procedure(fcbx) external;
```

```
 961     950 e 0F02                                    /* Drive may be password protected...
 962     951 e 0F02                                       Get passw from user and compare
 963     952 e 0F02                                       with Password in label.
 964     953 e 0F02                                       Label password is encoded by first
 965     954 e 0F02                                       reversing each char nibble and then
 966     955 e 0F02                                       XOR'ing with the sum of the pass.
 967     956 e 0F02                                       S2 in label = that sum.  */
 968     957 e 0F02
 969     958 e 0F02          declare
 970     959 e 0F0A                  passwd(8)       bit(8) based(passptr),
 971     960 e 0F0A
 972     961 e 0F0A                  passptr         pointer,
 973     962 e 0F0A                  convptr         pointer,
 974     963 e 0F0A                  pchar(8)        bit(8),
 975     964 e 0F0A                  cvpass(8)       char(1) based(convptr),
 976     965 e 0F0A                  inpass          char(8),
 977     966 e 0F0A                  (i,j,fcbx)      bin fixed(7);
 978     967 e 0F0A
 979     968 e 0F0A          labdone = true;
 980     969 e 0F0F
 981     970 e 0F0F          passptr = addr(dirm(fcbx).diskpass);
 982     971 e 0F27          convptr = addr(pchar(1));
 983     972 e 0F2D
 984     973 e 0F2D          do i = 1 to 8;                  /* XOR each character  */
 985     974 e 0F39                  pchar(i) = bool(passwd(i),dirm(fcbx).fs1,'0110'b);
 986     975 e 0F78          end;
 987     976 e 0F78
 988     977 e 0F78          if cvpass(8) <= ' ' then return; /* no password */
 989     978 e 0F93
 990     979 e 0F93          put skip(2) list('Directory is password protected.');
 991     980 e 0FAF          put skip list('Password, please.  >');
 992     981 e 0FCB          get list(inpass);
 993     982 e 0FE4          inpass = translate(inpass,UPPERCASE,LOWERCASE);
 994     983 e 100D
 995     984 e 100D          j = 8;
 996     985 e 1012          do i = 1 to 8;
 997     986 e 101E                  if substr(inpass,i,1) ^= cvpass(j) then call errprint(errpass);
 998     987 e 1062                  j = j - 1;
 999     988 c 106D          end;
1000     989 c 106D
1001     990 c 106D  end getpass;
1002     991 e 106D
1003     992 e 106D  collapse: procedure external;
1004     993 e 106D
1005     994 e 106D          declare whichbuf        bin fixed(7),
1006     995 e 1070                  enddcnt         bin fixed(15),
1007     996 e 1070                  (i,nout1,nout2) bin fixed(7);
1008     997 e 1070
1009     998 e 1070          dcnt = 0;
1010     999 e 1076          sect = 0;
1011    1000 e 107C          outdcnt = 0;
1012    1001 e 1082          whichbuf = 0;
1013    1002 e 1087          nout1 = 0;
1014    1003 e 108C          nout2 = 0;
1015    1004 e 1091          lastsect = 0;
1016    1005 e 1097          enddcnt = lastdcnt + nempty;
1017    1006 e 10A1          lastdcnt = 0;
1018    1007 e 10A7          bufptr1 = addr(outbuf(0));
1019    1008 e 10AD          bufptr2 = addr(buffer2(0));
1020    1009 e 10B3
```

```
1021    1010 e 1083         do while(dcnt <= enddcnt);              /* read up to last dcnt */
1022    1011 e 108F
1023    1012 e 108F                 call read_sector(dcnt,dirptr);
1024    1013 e 10C5
1025    1014 e 10C5                 do i = 0 to nfcbs while(dcnt <= enddcnt);
1026    1015 e 10F1                         if dir_fcb(i).user ^= '0E5'b4 &
1027    1016 e 1128                            dirm(i).user ^= SFCBmark then do;
1028    1017 e 1128
1029    1018 e 1128                                 if whichbuf = 0 then
1030    1019 e 112F                                         call fill(bufptr1,i,nout1,whichbuf);
1031    1020 e 1137                                  else call fill(bufptr2,i,nout2,whichbuf);
1032    1021 e 113D                         end;
1033    1022 e 113D                         dcnt = dcnt + 1;
1034    1023 e 1147                 end;
1035    1024 e 1147
1036    1025 e 1147                 sect = sect + 1;
1037    1026 e 114B                 if nout1 = nfcbs1 then call flush_write(nout1,bufptr1);
1038    1027 e 115D                 else if nout2 = nfcbs1 then call flush_write(nout2,bufptr2);
1039    1028 e 1170         end;
1040    1029 e 1170
1041    1030 e 1170         dcnt = dcnt - 1;                        /* fill unused slots in buffer
1042    1031 e 1174                                                    with empty...scratch rest of
1043    1032 e 1174                                                    dir */
1044    1033 e 1174         if whichbuf = 0 then call fill2(bufptr1,nout1);
1045    1034 e 1183         else call fill2(bufptr2,nout2);
1046    1035 c 118A
1047    1036 c 118A end collapse;
1048    1037 e 118A
1049    1038 e 118A fill: proc(bufptr,i,nout,whichbuf);
1050    1039 e 118A         declare bufptr          pointer,
1051    1040 e 1197                 (i,j,nout)      bin fixed(7),
1052    1041 e 1197                 whichbuf        bin fixed(7),
1053    1042 e 1197
1054    1043 e 1197                 1 buffer(0:127) based(bufptr),
1055    1044 e 1197                  2 out          char(32);
1056    1045 e 1197
1057    1046 e 1197         buffer(nout).out = infcb(i).rest;
1058    1047 e 11CA
1059    1048 e 11CA         lastdcnt = lastdcnt + 1;
1060    1049 e 11CE         nout = nout + 1;
1061    1050 e 11D4         if nout = nfcbs1 then whichbuf = mod((whichbuf + 1),2);
1062    1051 c 1202
1063    1052 c 1202 end fill;
1064    1053 e 1202
1065    1054 e 1202 flush_write: proc(nout,bufptr);
1066    1055 e 1202         declare nout            bin fixed(7),
1067    1056 e 120F                 bufptr          pointer;
1068    1057 e 120F
1069    1058 e 120F                                         /* always behind the read...thus don't
1070    1059 e 120F                                            need to test to see if read sector =
1071    1060 e 120F                                            write sector. */
1072    1061 e 120F         call write_sector(outdcnt,bufptr);
1073    1062 e 121B         outdcnt = outdcnt + nfcbs1;
1074    1063 e 1222         nout = 0;
1075    1064 e 1229         lastsect = lastsect + 1;
1076    1065 c 122E
1077    1066 c 122E end flush_write;
1078    1067 e 122E
1079    1068 e 122E fill2: proc(bufptr,nout);
1080    1069 e 122E
```

```
1081    1070 e 122E         declare (i,j,nout)      bin fixed(7),
1082    1071 e 123C                 bufptr          pointer,
1083    1072 e 123C                 1 buffer(0:127) based(bufptr),
1084    1073 e 123C                  2 user         bit(8),
1085    1074 e 123C                  2 rest(31)     bit(8);
1086    1075 e 123C
1087    1076 e 123C         do i = nout to nfcbs;
1088    1077 e 1254                 buffer(i).user = 'E5'b4;
1089    1078 e 1267                 do j = 1 to 31;
1090    1079 e 1273                         buffer(i).rest(j) = '00000000'b;
1091    1080 e 1298                 end;
1092    1081 e 1298         end;
1093    1082 e 1298
1094    1083 e 1298         lastdcnt = lastdcnt - 1;
1095    1084 e 129C         lasti = nout - 1;
1096    1085 e 12A7         call flush_write(nout,bufptr);
1097    1086 e 1289
1098    1087 e 1289         do i = 0 to nfcbs;                      /* prepare empty sector */
1099    1088 e 12CF                 buffer(i).user = 'E5'b4;
1100    1089 e 12E2                 do j = 1 to 31;
1101    1090 e 12EE                         buffer(i).rest(j) = '00000000'b;
1102    1091 e 1313                 end;
1103    1092 e 1313         end;
1104    1093 e 1313
1105    1094 e 1313                                             /* clear rest of directory */
1106    1095 e 1313         do while (outdcnt < dcnt);
1107    1096 e 131C                 call write_sector(outdcnt,bufptr);
1108    1097 e 1328                 outdcnt = outdcnt + nfcbs;
1109    1098 c 1332         end;
1110    1099 c 1332
1111    1100 c 1332 end fill2;
1112    1101 e 1332
1113    1102 e 1332 restore: procedure external;
1114    1103 e 1332         declare
1115    1104 e 1334                 1 xdpb          based(dpbp),
1116    1105 e 1334                  2 front        char(11),
1117    1106 e 1334                  2 chkvecb      bit(16);
1118    1107 e 1334
1119    1108 e 1334                                     /* if selected drive was permanent,
1120    1109 e 1334                                        then must force login of drive to
1121    1110 e 1334                                        restore good directory buffers and
1122    1111 e 1334                                        hash tables */
1123    1112 e 1334                     /* In CCP/M-86, this is done by setting
1124    1113 e 1334                        the login sequence number in the DPH
1125    1114 e 1334                        to zero, thus ensuring hard disks wi
1126    1115 e 1334                        also get reset.  Look at rtn 'seldsk'
1127    1116 e 1334                        in 'initdira.a86' */
1128    1117 e 1334 /*
1129    1118 e 1334         if chkvecb = '1000000000000000'b then do;
1130    1119 e 1334                 if drive = 0 then drive = curdisk;
1131    1120 e 1334                 else drive = drive - 1;
1132    1121 e 1334                 checked = 0;
1133    1122 e 1334                 call reset();
1134    1123 e 1334                 errorcode = select(drive);
1135    1124 e 1334                 chkvecb = '1000000000000000'b;
1136    1125 e 1334                 errorcode = select(drive);
1137    1126 e 1334         end;
1138    1127 e 1334 */
1139    1128 e 1334     call sysunlock();       /* unlock the disk system mhf 1/83 */
1140    1129 e 1337         dphp = seldsk(curdisk);         /* restore drive */
```

```
1141    1130 e 1341          call reset();                    /* reset disk system */
1142    1131 e 1344          errorcode = select(curdisk);
1143    1132 e 134E      call conunlock();       /* allow switching of consoles whf 1/83 */
1144    1133 e 1351
1145    1134 e 1351          call reboot;
1146    1135 c 1355
1147    1136 c 1355  end restore;
1148    1137 e 1355
1149    1138 e 1355                                  /* read logical record # to dma address */
1150    1139 e 1355  read_sector: procedure(lrcd,dmaaddr) external;
1151    1140 e 1355          dcl
1152    1141 e 1363             lrcd      bin fixed(15),
1153    1142 e 1363             prcd      decimal(7,0),
1154    1143 e 1363             dmaaddr   pointer;                 /* dma address */
1155    1144 e 1363
1156    1145 e 1363          prcd = lrcd/nfcbs1;
1157    1146 e 1378          gtrk = track(prcd);
1158    1147 e 1385          call settrk(gtrk);
1159    1148 e 138B          gsec = sector(prcd);
1160    1149 e 1395          call setsec(gsec);
1161    1150 e 139B
1162    1151 e 139B          call bstdma(dmaaddr);
1163    1152 e 13A7          if rdsec() ~= 0 then do;
1164    1153 e 13B1          put skip list('While reading record ',prcd);
1165    1154 e 13DF          put list(': track ',gtrk,', sector',gsec);
1166    1155 e 1419          call errprint(errXREAD);
1167    1156 c 142B      end;
1168    1157 c 142B
1169    1158 c 142B  end read_sector;
1170    1159 e 142B
1171    1160 e 142B
1172    1161 e 142B                                  /* write logical record # from dma address */
1173    1162 e 142B  write_sector: procedure(lrcd,dmaaddr) external;
1174    1163 e 142B          dcl
1175    1164 e 1439             lrcd          bin fixed(15),
1176    1165 e 1439             dmaaddr       pointer,             /* dma address */
1177    1166 e 1439             prcd          decimal(7,0);
1178    1167 e 1439
1179    1168 e 1439          prcd = lrcd/nfcbs1;                  /* #fcbs/phys rec */
1180    1169 e 1451          gtrk = track(prcd);
1181    1170 e 145B          call settrk(gtrk);
1182    1171 e 1461          gsec = sector(prcd);
1183    1172 e 146B          call setsec(gsec);
1184    1173 e 1471
1185    1174 e 1471          call bstdma(dmaaddr);
1186    1175 e 147D          if wrsec(1) ~= 0 then do;
1187    1176 e 1487          put skip list('While writing record ',prcd);
1188    1177 e 1485          put list(': track ',gtrk,', sector',gsec);
1189    1178 e 14EF          call errprint(errXREAD);
1190    1179 c 1501      end;
1191    1180 c 1501
1192    1181 c 1501
1193    1182 c 1501  end write_sector;
1194    1183 e 1501
1195    1184 e 1501
1196    1185 e 1501                                  /* select disk drive */
1197    1186 e 1501  dselect: procedure((d)) external;
1198    1187 e 1501          dcl
1199    1188 e 1518              p              ptr,
1200    1189 e 1518              wdalv(16)      fixed(15) based(p),
```

```
1201    1190 e 1518             btalv(16)       fixed(7)  based(p),
1202    1191 e 1518             all             bit(16),
1203    1192 e 1518             d               fixed(7);
1204    1193 e 1518
1205    1194 e 1518
1206    1195 e 1518         dcl
1207    1196 e 1518                 1 dpb based (dpbp),
1208    1197 e 1518                   2 sec bit(16),
1209    1198 e 1518                   2 bsh bit(8),
1210    1199 e 1518                   2 blm bit(8),
1211    1200 e 1518                   2 exm bit(8),
1212    1201 e 1518                   2 dsm bit(16),
1213    1202 e 1518                   2 drm bit(16),
1214    1203 e 1518                   2 al0 bit(8),
1215    1204 e 1518                   2 al1 bit(8),
1216    1205 e 1518                   2 cks bit(16),
1217    1206 e 1518                   2 off bit(8);
1218    1207 e 1518
1219    1208 e 1518         if d = 0 then d = curdsk();
1220    1209 e 1527         else d = d - 1;
1221    1210 e 152B
1222    1211 e 152B         errorcode = select(d);                  /* sync BIOS & BDOS */
1223    1212 e 1535         dphp = seldsk(d);
1224    1213 e 153F         if dphp = null then call errprint(errBIOS);/* can't select disk */
1225    1214 e 1561
1226    1215 e 1561         xlt = xlt1;
1227    1216 e 156A         dpbp = dpbptr;
1228    1217 e 1575
1229    1218 e 1575         dspt = decimal(spt);    /**** whf 1/8/83 ****/
1230    1219 c 1587
1231    1220 c 1587 end dselect;
1232    1221 e 1587
1233    1222 e 1587                                 /* convert logical rcd # to physical sector */
1234    1223 e 1587 sector: procedure(i) returns(fixed(15)) external;
1235    1224 e 1587         dcl
1236    1225 e 158E             i    decimal(7,0);
1237    1226 e 158E
1238    1227 e 158E         return(sectrn(binary(mod(i,dspt),15),xlt));
1239    1228 c 15B2
1240    1229 c 15B2 end sector;
1241    1230 e 15B2
1242    1231 e 15B2
1243    1232 e 15B2                                 /* logical record # to physical track */
1244    1233 e 15B2 track: procedure(i) returns(fixed(15)) external;
1245    1234 e 15B2         dcl
1246    1235 a 15B9             i    decimal(7,0);
1247    1236 a 15B9
1248    1237 a 15B9         return(offset + binary(i/dspt,15));
1249    1238 c 15E5
1250    1239 c 15E5 end track;
1251    1240 e 15E5
1252    1241 e 15E5
1253    1242 e 15E5                                 /* logical record # to physical block */
1254    1243 e 15E5 conv: procedure(i) returns(fixed(15)) external;
1255    1244 e 15E5         dcl
1256    1245 e 15EC             i       fixed(7),
1257    1246 e 15EC             j       fixed(15),
1258    1247 e 15EC             p       ptr,
1259    1248 e 15EC             n       fixed(7) based(p);
1260    1249 e 15EC
```

```
1261    1250 e 15EC          p = addr(j);
1262    1251 e 15F2          j = 0;
1263    1252 e 15F8          n = i;
1264    1253 e 1604          return(j);
1265    1254 c 1609 end conv;
1266    1255 e 1609
1267    1256 e 1609 patch: procedure;
1268    1257 e 1609      dcl i fixed(15);
1269    1258 e 160C
1270    1259 e 160C      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1271    1260 e 1634      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1272    1261 e 165C      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1273    1262 e 1684      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1274    1263 e 16AC      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1275    1264 e 16D4      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1276    1265 e 16FC      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1277    1266 e 1724      i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;  i=i+5;
1278    1267 c 174D end patch;
1279    1268 a 174D
1280    1269 a 174D
1281    1270 a 174D
1282    1271 a 174D end initdir;Code Size: 1750
1283    Data Size: 3980
1284    End of Compilation
```



```
**** CROSS REFERENCE *****
          addr  411, 494, 499, 981, 982,1018,1019,1261
           al0 1214
           al1 1215
           all 1202
        allvec   75
        allwds  692, 696
       allxfcb  415, 685, 700
        alvptr  200
      askclear  250, 714
         asker  711, 714, 931, 942
           bin  184, 231, 232, 233, 234, 238, 309, 310, 311, 312, 313, 314, 315,
                316, 317, 318, 319, 320, 321, 322, 323, 326, 347, 369, 428, 445,
                463, 558, 564, 600, 671, 693, 729, 783, 828, 947, 977,1005,1006,
               1007,1051,1052,1066,1081,1152,1175
        binary 1238,1248
           bit   54,  55,  56,  59,  60,  62,  63,  64,  65,  66,  67,  68,  69,
                 70,  71,  72,  77,  82,  83,  86,  87, 132, 140, 141, 147, 152,
                158, 168, 175, 185, 186, 195, 196, 197, 217, 227, 228, 240, 329,
                330, 331, 332, 333, 334, 335, 336, 339, 561, 872, 931, 970, 974,
               1084,1085,1117,1202,1208,1209,1210,1211,1212,1213,1214,1215,1216,
               1217
        blkmsk  213
          blks  227
       blkshft  212
           blm 1210
          bool  985
         break   59
           bsh 1209
        bstdma  116,1162,1185
         btalv 1201
        buffer 1054,1057,1083,1088,1090,1099,1101
       buffer2  139, 494,1019
        bufptr 1049,1050,1054,1065,1067,1072,1079,1082,1083,1096,1107
```

```
      fill2 1044,1045,1079,1111
     filsiz   84
    findxfcb  831, 872, 892
      flush 1037,1038,1065,1077,1096
      fname  182, 733
        frc  187
      frest  160
      front 1116
        fs1  185, 985
        fs2  186
      ftype  183, 734
        get  397, 448, 936, 992
    getdisk  389, 444, 454
     getdpb   79
      getio   55
    getpass  578, 646, 960,1001
     getusr   80
    getxfcb  641, 670, 682, 848
       gsec  316,1159,1160,1165,1182,1183,1188
       gtrk  316,1157,1158,1165,1180,1181,1188
       hash  203
          i  369, 388, 389, 427, 428, 444, 445, 448, 449, 450, 452, 463, 558,
             574, 577, 578, 579, 583, 584, 600, 635, 636, 637, 641, 646, 651,
             670, 671, 676, 677, 678, 729, 783, 800, 801, 802, 805, 810, 812,
             828, 846, 847, 848, 947, 949, 950, 951, 954, 977, 984, 985, 996,
             997,1007,1025,1026,1027,1030,1031,1049,1051,1057,1081,1087,1088,
            1090,1098,1099,1101,1234,1236,1238,1244,1246,1248,1254,1256,1263,
            1268,1270,1271,1272,1273,1274,1275,1276,1277
        idx  872, 877, 881, 882, 884
         in    8,  10
      index  452
      infcb  162, 744,1057
     infcb2  165, 676, 677, 678, 884
       init  175, 252, 421, 460, 547
    initdir    1,  12,1282
    initmsg  245, 396
     inpass  976, 992, 993, 997
          j  428, 463, 558, 729, 783, 804, 805, 898, 901, 902, 909, 911, 947,
             953, 954, 977, 995, 997, 998,1051,1081,1089,1090,1100,1101,1257,
            1261,1262,1264
      junk1  169, 560
          k  463, 729, 740, 741, 742, 744, 755, 761, 781, 783, 802, 805, 828,
             831, 844, 846, 851, 862, 863, 864, 895, 898, 900, 901
          l    4, 463
    labdone  334, 576, 580, 979
    lastdcnt 312, 470, 474, 475, 477, 478, 479, 486, 487, 491, 519, 653, 666,
             843,1016,1017,1059,1094
      lasti  313, 471, 524, 651, 844,1095
    lastsect 314, 506, 652,1015,1075
    lastsfcb 311, 474, 475, 478, 479, 486, 487, 666
      lastx  313, 465, 884
       list    4, 397, 430, 447, 448, 707, 935, 936, 990, 991, 992,1164,1165,
            1187,1188
     logvec   72
  lowercase  261, 449, 993
       lrcd 1150,1152,1156,1173,1175,1179
       lyes  256, 398, 938
       main   12
       make   70
```

```
        mask  193, 210, 900
      maxsave  231, 673, 697, 698
       maxwds  691, 695
       mediaf  196
       memptr   40
       memsiz   41
       memwds   42
         mode  158, 910
     morexfcb  544, 822, 870
          msg  919, 921, 923, 931, 933, 935
            n 1259,1263
         name  167, 239, 676, 786, 791, 805, 884
       nempty  309, 477, 512, 617, 655, 658,1016
      newdcnt  319, 470, 475, 476, 479, 481, 487, 488, 491, 520, 768, 769
     newlasti  322, 471, 491, 523
        nfcbs  310, 405, 406, 536, 574, 635, 770, 846, 862, 881, 949,1025,1087,
               1098
       nfcbs1  310, 406, 491, 508, 766, 769,1037,1038,1061,1073,1108,1156,1179
           no    8,  10
     notsaved  234, 543, 620, 680, 837
         nout 1049,1051,1057,1060,1061,1065,1066,1074,1079,1081,1087,1095,1096
        nout1 1007,1013,1030,1037,1044
        nout2 1007,1014,1031,1038,1045
         null 1224
         nxfcb  233, 465, 619, 673, 674, 675, 676, 677, 678, 752, 800, 811, 840
           of    1,1284
          off 1217
       offset  219,1248
      olddcnt  564, 573, 589
         open   63
      openvec  119, 376
           ot  731
          out 1055,1057
         outb  143, 744
        outb2  146, 747, 950
        outb3  150, 748, 951, 954
        outb4  154, 908, 910
       outbuf  135, 499,1018
      outdcnt  318, 858, 867,1011,1072,1073,1106,1107,1108
       outidx  320, 523, 744, 747, 748, 758, 759, 764, 770, 771, 863, 906, 907
       outptr  143, 146, 150, 154, 302, 494, 498, 499, 507, 541, 731, 768
         outx  806, 814
            p 1199,1200,1201,1258,1259,1261
         pass    8,  10
        pass1  264
      passptr  970, 972, 981
       passwd  970, 985
        patch 1267,1278
        pchar  974, 982, 985
       phymsk  221, 405
       physhf  220
        pmode  168, 240, 677, 911
         prcd 1153,1156,1157,1159,1164,1177,1179,1180,1182,1187
        print    3
         proc 1049,1065,1079
      program    4
          ptr   40,  43,  44,  45,  57,  58,  63,  64,  65,  67,  68,  69,  70,
                71,  74,  75,  78,  79,  82,  83,  84,  85,  87, 110, 115, 116,
               192, 194, 198, 199, 200, 201, 202, 203, 206, 209,1199,1258
```

```
      ptreos  338, 340
     putxfcb  755, 864, 895, 916
       query  416, 703, 726
       rdbuf   58
       rdcon   47
       rdran   82
       rdrdr   49
       rdsec  113,1163
       rdseq   68
      rdstat   54
        read  413, 521, 535, 571, 663, 852, 857, 880,1023,1150,1169
    readonly  336, 400, 401
      reboot   46, 398,1145
     recover  252, 711
        redo  329, 469, 616, 666, 712, 758, 771
      rename   71
     replace  125
      resdrv   86
       reset   61,1141
        rest  133, 137, 141, 144, 148, 152, 159, 163, 173, 189, 562, 735, 744,
              748, 787, 792, 954,1057,1085,1090,1101
     restore  422, 722, 926,1113,1147
       rovec   77, 400
           s    8,  10
         scb  351
        sear   65
       searn   66
         sec 1208
        sect  317, 466, 506, 509, 618, 652, 662,1010,1036
      sector  413, 507, 521, 535, 541, 571, 589, 663, 767, 768, 852, 857, 867,
              880,1023,1072,1107,1150,1159,1169,1173,1182,1193,1234,1240
      sectrn  115,1238
      seldsk  110,1140,1223
      select   62,1142,1222
      setdma   74
       seter  353, 379
       setio   56
      setrec   85
      setsec  112,1160,1183
      settrk  111,1158,1181
      setusr   81
        sfcb  156, 908, 910
     sfcbidx  323, 907, 908, 910
       sfcbm  155
    sfcbmark  325, 416, 583, 632, 746, 951,1027
    sfcboffs  323, 906, 907, 908, 910
        size  693, 695, 696, 697,1282,1283
       soffs  352, 378
      source    4
      space1  195
      space2  197
         spt  211,1229
       stamp  170, 241, 678, 909
      stamps  157, 908
       strip  552, 592, 719
     syslock  120, 409
   sysunlock  121,1139
        term  268, 923
       track 1157,1180,1244,1250
```

```
     translate  449, 993
          true  126, 580, 621, 712, 715, 885, 912, 940, 979
     uppercase  259, 449, 993
          user  132, 136, 140, 147, 151, 166, 181, 238, 416, 577, 583, 584, 632,
                636, 637, 675, 732, 742, 747, 785, 790, 801, 802, 810, 847, 864,
                882, 950, 951,1026,1027,1084,1088,1099
       usernum  324, 637, 640, 646, 648, 655, 675, 742, 746, 751, 847, 848, 882,
                883
          vers   60, 371
         wdalv 1200
      whichbuf 1005,1012,1029,1030,1031,1044,1049,1052,1061
        wpdisk   76
         wrcon   68
         write  507, 541, 589, 767, 768, 867,1037,1038,1065,1072,1077,1096,1107,
               1173,1193
     writeflag  331, 859, 867, 912
         wrlst   51
     wrongdisk  388, 427, 442
         wrpun   50
         wrran   83
        wrranz   87
         wrsec  114,1186
         wrseq   69
         wrstr   57
          xdpb 1115
         xfcb2  789, 805
         xfcbs  237, 675, 676, 677, 678, 801, 802, 810, 884, 909, 911
           xlt  206,1226,1238
          xlt1  194,1226
          xptr  235, 237, 696, 789
           yes  255, 398, 938
         yesno  257, 397, 398, 936, 938
            yn  254, 396, 935
        zeroes  175, 748
**** end cross reference ***
```

```
                        ; ************************************************************************
                        ; initdira - Provides BIOS (XIOS) assembler interface for INITDIR.PLI
                        ;         - Also provides examples of how to:
                        ;             1. Call CCP/M-86 XIOS
                        ;             2. Lock up disk system for direct disk I/O
                        ;             3. Lock up console, to prevent a job from being switched out
                        ; ************************************************************************

                                cseg

                                public  bstdma          ; sets DMA segment and offset
                                public  rdsec           ; reads a physical sector
                                public  sectrn          ; translates a sector
                                public  seldsk          ; selects a drive
                                public  setsec          ; sets the sector to be read/written
                                public  settrk          ; sets the track to be read/written
                                public  wrsec           ; writes a physical sector

                                public  openvec         ; returns open files vector
                                public  syslock         ; locks up the disk system
                                public  sysunlock       ; unlocks the disk system
                                public  conlock         ; locks the console into foreground mode
                                public  conunlock       ; unclocks the console

0009                    IO_SELDSK equ     9             ; XIOS function number
000A                    IO_READ   equ    10             ; XIOS function number
000B                    IO_WRITE  equ    11             ; XIOS function number
0091                    P_PRIORITY equ  145             ; BDOS function: set Process PRIORITY
009A                    S_SYSDAT  equ   154             ; BDOS function: get SYStem DATa page address
009C                    P_PDADR   equ   156             ; BDOS function: get Process Descriptor address
0087                    Q_OPEN    equ   135             ; Open Queue
008A                    Q_READC   equ   138             ; Read Queue Conditional
008B                    Q_WRITEC  equ   139             ; Write Queue Conditional

0028                    XIOS_ptr  equ   dword ptr .28h  ; loc of XIOS entry in SYSDAT
0088                    OPVEC     equ   word ptr .88h   ; loc of Open_Files_on_Drives vector
0010                    UDA_seg   equ   word ptr 10h    ; loc of UDA seg in Process Descriptor


                        ;************************************************************
                        ;***   PL/I Utility Functions                             ***
                        ;************************************************************
                        getp1:  ; get single byte parameter to register DL
0000 8B1F                       mov     bx,[bx]         ;BX = .char
0002 8A17                       mov     dl,[bx]         ;to register DL
0004 C3                         ret

                        getp2:  ; get single word value to DX
                        getp2i: ; same as getp2
```

```
                                mov     bx,[bx]
                                mov     dx,[bx]
                                ret


                        getsu:  ;get sysdat and uda addrs
                                ;enters: DS=local data seg
                                ;exits:  DS=SYSDAT seg, ES=UDA seg (for call to XIOS)
000A 8B0E0200     R             mov     cx,udaaddr      ;get the saved value
000E 0BC9                       or      cx,cx           ;set flags
0010 7409           001B        jz      getsul          ;uninitialized, go do the OS call
0012 8EC1                       mov     es,cx           ;we've been here before, just load regs
0014 8B0E0000     R             mov     cx,sysaddr
0018 8ED9                       mov     ds,cx
001A C3                         ret

                        getsul:
001B B19C                       mov     cl,P_PDADR      ;get Process Descriptor
001D CDE0                       int     0E0h            ;call BDOS
001F 26884F10                   mov     cx,es:UDA_seg[bx] ;grab UDA_seg
0023 890E0200     R             mov     udaaddr,cx      ;save for future calls
0027 51                         push    cx              ;save UDA_seg
0028 B19A                         mov   cl,S_SYSDAT     ;get address of SYStem DATa area
002A CDE0                         int   0E0h            ;call BDOS
002C 8CC1                         mov   cx,es           ;mov ds,es
002E 890E0000     R               mov   sysaddr,cx      ;save for future calls
0032 8ED9                         mov   ds,cx
0034 07                         pop     es              ;restore UDA_seg
0035 C3                         ret


                        ;*****************************************************************
                        ;***   Simulate old XIOS style functions                       ***
                        ;*****************************************************************
                        bstdma: ; sets DMA segment and offset
0036 E8CCFF         0005        call    getp2           ;dma address to DX
0039 89163100     R             mov     dmaoff,dx       ;stuff addr in IOPB's offset field
003D 8C1E2F00     R             mov     dmaseg,ds       ;assume all addresses relative to DS
0041 C3                         ret                     ;no BDOS/XIOS call, just init the IOPB


                        sectrn: ; translates a sector
0042 E8C0FF         0005        call    getp2           ;get sector number to DX
0045 8BDA                       mov     bx,dx           ;no translation: return (unchanged) value
0047 C3                         ret


                        setsec: ; sets the sector to be read/written
0048 E8BAFF         0005        call    getp2i          ;sector number to DX
004B 89162D00     R             mov     sector,dx       ;stuff sector into IOPB
004F C3                         ret
```

```
                                settrk: ; sets the track to be read/written
0050 E882FF             0005            call    getp2i          ;track number to DX
0053 89162B00       R                   mov     track,dx        ;stuff track into IOPB
0057 C3                                 ret


                                ;*****************************************************************
                                ;***  Physical I/O calls                                       ***
                                ;*****************************************************************
                                rdsec:  ; reads a physical sector
0058 B80A00                             mov     ax,IO_READ
005B E90300             0061            jmp     xiosiopb        ;jump around this code

                                wrsec:  ; writes a physical sector
005E B80B00                             mov     ax,IO_WRITE     ;fall thru to xiosiopb

                                xiosiopb:       ;put the IOPB on the stack, call XIOS
0061 1E                                 push    ds              ;ds will contain SYSDAT seg
0062 06                                 push    es              ;es will contain UDA seg
                                                ;someday change this to a block move?
0063 8A2E2900       R                   mov     ch,mscnt
0067 8A0E2A00       R                   mov     cl,drv
006B 51                                 push    cx              ;1st word of IOPB
006C 8B0E2B00       R                   mov     cx,track
0070 51                                 push    cx              ;2nd word
0071 8B0E2D00       R                   mov     cx,sector
0075 51                                 push    cx              ;3rd word
0076 8B0E2F00       R                   mov     cx,dmaseg
007A 51                                 push    cx              ;4th word
007B 8B0E3100       R                   mov     cx,dmaoff
007F 51                                 push    cx              ;5th word

0080 50                                 push    ax              ;save XIOS_function
0081 E886FF             000A            call    getsu           ;set up DS-SYSDAT and ES-UDA
0084 58                                 pop     ax              ;restore XIOS_function
0085 FF1E2800                           callf   XIOS_ptr        ;call indirect the XIOS
                                                                ;bl contains return status
0089 59                                 pop     cx              ;dma offset
008A 59                                 pop     cx              ;dma segment
008B 59                                 pop     cx              ;track
008C 59                                 pop     cx              ;sector
008D 59                                 pop     cx              ;drv & mscnt
008E 07                                 pop     es              ;restore original es
008F 1F                                 pop     ds              ;ditto for ds
0090 C3                                 ret


                                ;*****************************************************************
                                ;*** XIOS Select Disk Routine                                   ***
                                ;*****************************************************************
```

```
                        seldsk: ; selects a drive
                                ; also resets login sequence number of drive to 0,
                                ;   to force permanent media to be logged in again on disk reset

0091 E86CFF          0000       call    getpl           ;drive to DL
0094 88162A00    R             mov     drv,dl          ;stuff drive into IOPB
0098 061E                      push es ! push ds       ;save context ************
                                                       ;do the XIOS SELDSK call
009A 52                        push    dx              ;save drive
009B E86CFF          000A      call    getsu           ;set up DS and ES
009E 59                        pop     cx              ;restore drive
009F B80900                    mov     ax,IO_SELDSK
00A2 BA0000                    mov     dx,0            ;this better not be the first call
00A5 FF1E2800                  callf   XIOS_ptr        ;call indirect XIOS
                                                       ;xfer DPH locally
00A9 0706                      pop es ! push es        ;restore & save Data Segment into es
00AB BF0400      R             mov     di,offset dph   ;set up destination
00AE 8BF3                      mov     si,bx           ;ptr to dph returned from XIOS call
00B0 C6440600                  mov     log_seqn[si],0  ;force disk reset: 0 login sequence number
00B4 B91400                    mov     cx,dphsiz
00B7 F3A4                      rep movsb               ;move copy of DPH into local storage
                                                       ;xfer DPB locally
00B9 BF1800      R             mov     di,offset dpb   ;set up destination
00BC 2688360C00  R             mov     si,es:dpbptr    ;get this info from DPH
00C1 B91100                    mov     cx,dpbsiz
00C4 F3A4                      rep movsb               ;move copy of DPB into local storage
                                                       ;cleanup
00C6 1F07                      pop ds ! pop es         ;restore context ************
00C8 C7060C001800 R            mov     dpbptr,offset dpb ;set up local ptr in DPH
00CE B80400      R             mov     bx,offset dph   ;return address of local copy of DPH
00D1 C3                        ret


                        ;*****************************************************************
                        ;***  Open Drives Vector                                       ***
                        ;*****************************************************************
                        openvec:        ; returns vector of drives with open files
00D2 06                        push    es              ;save extra seg
00D3 1E                        push    ds              ;save data seg
00D4 B19A                      mov     cl,S_SYSDAT     ;look in the system data page
00D6 CDE0                      int     0E0h            ;call bdos
00D8 26A18800                  mov     ax,es:OPVEC     ;get the vector of drives containing open files
00DC 8BD8                      mov     bx,ax           ;stuff both regs
00DE 1F                        pop     ds              ;restore data seg
00DF 07                        pop     es              ;restore extra seg
00E0 C3                        ret


                        ;*****************************************************************
                        ;***  System Lock and Unlock Routines                          ***
                        ;*****************************************************************
                        syslock:        ; locks up the disk system
```

```
                                        ; returns 0 in ax,bx if everything ok, -1 o.w.
00E1 06                         push    es              ;save extra seg
00E2 1E                         push    ds              ;save data seg

00E3 B19A                       mov     cl,S_SYSDAT     ;look in the system data page
00E5 CDE0                       int     0E0h            ;call bdos
00E7 268B0E8800                 mov     cx,es:OPVEC     ;get the vector of drives containing open files
00EC F7C1FFFF                   test    cx,0FFFFh       ;check all drives
00F0 7539           012B        jnz     syslfail        ;fail if any open files

00F2 B98700                     mov     cx,Q_OPEN
00F5 BA3300         R           mov     dx,offset qpb   ;mxdisk queue parm block
00F8 CDE0                       int     0E0h            ;call bdos
00FA 0BC0                       or      ax,ax           ;test return
00FC 750F           010D        jnz     sysltry2        ;if non zero, try kluge
00FE B98A00                     mov     cx,Q_READC      ;see if we can read the queue
0101 BA3300         R           mov     dx,offset qpb   ;insurance
0104 CDE0                       int     0E0h            ;call bdos
0106 0BC0                       or      ax,ax           ;test retrun
0108 7521           012B        jnz     syslfail        ;fail if we can't read mxdisk queue
010A E91800         0125        jmp     syslokay        ;okay, tell 'em so

                    sysltry2:                           ;kluge for old systems
010D B98700                     mov     cx,Q_OPEN
0110 BA4300         R           mov     dx,offset qpb2  ;mxdisk queue parm block
0113 CDE0                       int     0E0h            ;call bdos
0115 0BC0                       or      ax,ax           ;test return
0117 7512           012B        jnz     syslfail        ;if non zero
0119 B98A00                     mov     cx,Q_READC      ;see if we can read the queue
011C BA4300         R           mov     dx,offset qpb2  ;insurance
011F CDE0                       int     0E0h            ;call bdos
0121 0BC0                       or      ax,ax           ;test retrun
0123 7506           012B        jnz     syslfail        ;fail if we can't read mxdisk queue
                    syslokay:
0125 B80000E90300   012E        mov ax,0 ! jmp syslret  ;return code 0, everything okay
                    syslfail:
012B B8FFFF                     mov ax,0FFFFh           ;return code -1, failure
                    syslret:
012E 8BD8                       mov     bx,ax           ;stuff both regs
0130 1F                         pop     ds              ;restore data seg
0131 07                         pop     es              ;restore extra seg
0132 C3                         ret


                    sysunlock:      ;undoes a 'syslock' function
0133 B98B00                     mov     cx,Q_WRITEC     ;conditionally write to mxdisk queue
0136 BA3300         R           mov     dx,offset qpb
0139 CDE0                       int     0E0h
013B B98B00                     mov     cx,Q_WRITEC     ;kluge to handle old systems
013E BA4300         R           mov     dx,offset qpb2
0141 CDE0                       int     0E0h
```

```
                                ret


                        ;*****************************************************************
                        ;***  Console Lock and Unlock Routines                         ***
                        ;*****************************************************************

0002                    CCB_BACKGRD  equ 0002h          ;Console in Background mode
0008                    CCB_NOSWITCH equ 0008h          ;Console not allowed to switch mode
002C                    CCB_SIZE     equ 2Ch            ;size of CCB
000E                    CCB_STATE    equ word ptr 0Eh   ;addr of Console State word in CCB
0020                    CON_NUM      equ byte ptr 020h  ;addr of console number in PD
0054                    CCB_PTR      equ word ptr .054h ;addr of CCB table in SYSDAT

0006                    PD_FLAG      equ word ptr 06    ;addr of FLAGs word in PD
0002                    PD_KEEP      equ 0002h          ;Keep Process Flag
0097                    TOP_PRIOR    equ 151            ;What we set to for Top Priority (arbitrary)
00C8                    REG_PRIOR    equ 200            ;Default Regular Priority

                        conlock:        ;locks the console into the foreground, sets Keep Flag,
                                        ;  and boosts priority
                                        ;returns 0 in ax,dx if everything okay, -1 o.w.
0144 06                         push    es              ;save extra seg
0145 E84700             018F    call    concalc         ;ES=SYSDAT, bx=CCB offset
0148 26F7470E0200               test    es:CCB_STATE[bx],CCB_BACKGRD    ;is console in background?
014E 751C               016C    jnz     conlfail        ;background operation not allowed!
0150 26814F0E0800               or      es:CCB_STATE[bx],CCB_NOSWITCH   ;make sure they don't switch
0156 B19C                       mov     cl,P_PDADR      ;get Process Descriptor
0158 CDE0                       int     0E0h            ;call BDOS
015A 26814F060200               or      es:PD_FLAG[bx],PD_KEEP ;set Keep flag
0160 B297                       mov     dl,TOP_PRIOR    ;let's be quick
0162 B191                       mov     cl,P_PRIORITY   ;set Priority
0164 CDE0                       int     0E0h            ;call BDOS

                        conlokay:
0166 B80000E90300       016F    mov     ax,0 ! jmp conlret
                        conlfail:
016C B8FFFF                     mov     ax,0FFFFh
                        conlret:
016F 8BD8                       mov     bx,ax
0171 07                         pop     es              ;restore extra seg
0172 C3                         ret


                        conunlock:      ;undoes the 'conlock' function
0173 06                         push    es              ;save extra seg
0174 E81800             018F    call    concalc         ;ES=sysdat, bx=CCB offset
0177 2681670EF7FF               and     es:CCB_STATE[bx],NOT CCB_NOSWITCH ;let them switch now
017D B19C                       mov     cl,P_PDADR      ;get Process Descriptor
017F CDE0                       int     0E0h            ;call BDOS
```

```
                                and     es:PD_FLAGS[bx],NOT PD_KEEP ;reset Keep flag
0181 26816706FDFF               mov     dl,REG_PRIOR    ;finished with quick
0187 B2C8                       mov     cl,P_PRIORITY   ;set Priority
0189 B191                       int     0E0h            ;call BDOS
018B CDE0                       pop     es              ;restore extra seg
018D 07                         ret
018E C3


                        concalc:        ;put SYSDAT in ES, offset of CCB in bx
018F B99C00                     mov     cx,P_PDADR      ;get Process Descriptor Addr
0192 CDE0                       int     0E0h            ;call the bdos
0194 33C0                       xor     ax,ax           ;clear ax
0196 268A4720                   mov     al,es:CON_NUM[bx] ;grab the console number
019A B92C00                     mov     cx,CCB_SIZE     ;stuff cx with size
019D F7E1                       mul     cx              ;compute ccb address
019F 50                         push    ax              ;save ccb offset
01A0 B19A                       mov     cl,S_SYSDAT     ;get the System Data Segment
01A2 CDE0                       int     0E0h            ;call the bdos
01A4 5B                         pop     bx              ;restore ccb offset
01A5 26031E5400                 add     bx,es:CCB_PTR   ;compute offset of ccb
01AA C3                         ret


                        ;*****************************************************************
                        ;***  Data Segment                                             ***
                        ;*****************************************************************

                                dseg

0000 0000               sysaddr dw      0       ; save location for sysdat addr
0002 0000               udaaddr dw      0       ; save location for process uda addr

  0014                  dphsiz  equ     014h    ; size of Disk Parm Header
0004                    dph     rb      dphsiz  ; local save area
  0006                  log_seqn equ    byte ptr 6      ; byte to force reset of permanent media
  000C                  dpbptr  equ     word ptr dph+8  ; word of interest: DPB offset

  0011                  dpbsiz  equ     011h    ; size of Disk Parameter Buffer
0018                    dpb     rb      dpbsiz  ; local save area


0029                    iopb    rb      0       ; the iopb structure filled in by above rtns
0029 01                 mscnt   db      1       ; multi sector count
002A                    drv     rb      1       ; select drive
002B                    track   rw      1       ; select track
002D                    sector  rw      1       ; select sector
002F                    dmaseg  rw      1       ; set dma address
0031                    dmaoff  rw      1       ; set dma address
  000A                  iopbsiz equ     (offset $)-(offset iopb)
```

```
                                ;
                                ;
                                ;
0033                    qpb     rb      0       ; queue parameter block
0033 0000                       dw      0       ; reserved
0035 0000                       dw      0       ; queueid
0037 0000                       dw      0       ; nmsgs
0039 0000                       dw      0       ; buffer
003B 4D5864697368               db      'MXdisk  ' ; queue name
     2020
                                ;
0043                    qpb2    rb      0       ; queue parameter block number 2: be persistent
0043 0000                       dw      0       ; reserved
0045 0000                       dw      0       ; queueid
0047 0000                       dw      0       ; nmsgs
0049 0000                       dw      0       ; buffer
004B 6D7864697368               db      'mxdisk  ' ; queue name
     2020
                                ;
                                ;
END OF ASSEMBLY.  NUMBER OF ERRORS:  0.  USE FACTOR:  2%
```

BSTDMA	0036	L	13	90#							
CCB?BACKGRD	0002	N									
CCB?NOSWITCH	0008	N									
CCB?PTR	0054	V									
CCB?SIZE	002C	N									
CCB?STATE	000E	N									
CONCALC	018F	L	291	315	330#						
CONLFAIL	016C	L	293	304#							
CONLOCK	0144	L	24	287#							
CONLOKAY	0166	L	302#								
CONLRET	016F	L	303	306#							
CONUNLOCK	0173	L	25	313#							
CON?NUM	0020	N									
CS	SREG	V									
DMAOFF	0031	V	92	139	370#						
DMASEG	002F	V	93	137	369#						
DPB	0018	V	183	189	361#						
DPBPTR	000C	V	184	189	358#						
DPBSIZ	0011	N	185	360#	361						
DPH	0004	V	177	190	356#	358					
DPHSIZ	0014	N	180	355#	356						
DRV	002A	V	131	166	366#						
DS	SREG	V	68	81	93	127	153	167	188	199	204	216
			253									
ES	SREG	V	66	74	79	82	128	152	167	176	176	184
			188	198	202	205	215	220	254	290	292	294
			297	308	314	316	320	325	334	341		
GETP1	0000	L	47#	165								
GETP2	0005	L	52#	91	98							
GETP21	0005	L	53#	104	111							
GETSU	000A	L	60#	143	170							
GETSU1	001B	L	65	71#								
IOPB	0029	V	364#	371								
IOPBSIZ	000A	N	371#									
IO?READ	000A	N										
IO?SELDSK	0009	N										
IO?WRITE	0008	N										
LOG?SEQN	0006	N										
MSCNT	0029	V	130	365#								
OPENVEC	0002	L	21	197#								
OPVEC	0088	V	39#	202	220							
PD?FLAG	0006	N										
PD?KEEP	0002	N										
P?PDADR	009C	N										
P?PRIORITY	0091	N										
QPB	0033	V	225	230	261	375#						
QPB2	0043	V	238	243	264	383#						
Q?OPEN	0087	N										
Q?READC	008A	N										
Q?WRITEC	008B	N										
RDSEC	0058	L	14	119#								
REG?PRIOR	00C8	N										
SECTOR	002D	V	105	135	368#							

SECTRN	0042	L	15	97#			
SELDSK	0091	L	16	161#			
SETSEC	0048	L	17	103#			
SETTRK	0050	L	18	110#			
SS	SREG	V					
SYSADDR	0000	V	67	80	352#		
SYSLFAIL	012B	L	222	233	241	246	249#
SYSLOCK	00E1	L	22	212#			
SYSLOKAY	0125	L	234	247#			
SYSLRET	012E	L	248	251#			
SYSLTRY2	010D	L	228	236#			
SYSUNLOCK	0133	L	23	259#			
S?SYSDAT	009A	N					
TOP?PRIOR	0097	N					
TRACK	002B	V	112	133	367#		
UDAADDR	0002	V	63	75	353#		
UDA?SEG	0010	N					
WRSEC	005E	L	19	123#			
XIOSIOPB	0061	L	121	126#			
XIOS?PTR	0028	V					