ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE STAT
OBJECT MODULE PLACED IN :F1:STAT.OBJ
COMPILER INVOKED BY: PLM80 :F1:STAT.PLM DEBUG OPTIMIZE PAGEWIDTH(80)

```
  1          stat:
             do;
  2   1      declare
                 cpmversion literally '20h'; /* requires 2.0 cp/m */
              /* c p / m   s t a t u s   c o m m a n d  (s t a t) */

             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */
             /* status   status   status   status   status   status */

             /*
                     copyright(c) 1975, 1976, 1977, 1978, 1979
                     digital research
                     box 579
                     pacific grove, ca
                     93950
               */

             /* modified 10/30/78 to fix the space computation */
             /* modified 01/28/79 to remove despool dependencies */
             /* modified 07/26/79 to operate under cp/m 2.0 */

  3   1      declare jump byte data(0c3h),
                 jadr address data (.status);
                 /* jump to status */

             /* function call 32 returns the address of the disk parameter
             block for the currently selected disk, which consists of:
                     scptrk      (2 by) number of sectors per track
                     blkshf      (1 by) log2 of blocksize (2**blkshf=blksize)
                     blkmsk      (1 by) 2**blkshf-1
```

```
                    extmsk      (1 by) logical/physical extents
                    maxall      (2 by) max alloc number
                    dirmax      (2 by) size of directory-1
                    dirblk      (2 by) reservation bits for directory
                    chksiz      (2 by) size of checksum vector
                    offset      (2 by) offset for operating system
                */

 4  1      declare
            /* fixed locations for cp/m */
            bdosa literally '0006h',     /* bdos base */
            buffa literally '0080h',     /* default buffer */
            fcba  literally '005ch',     /* default file control block */
            dolla literally '006dh',     /* dollar sign position */
            parma literally '006eh',     /* parameter, if sent */
            rreca literally '007dh',     /* random record 7d,7e,7f */
            rreco literally '007fh',     /* high byte of random overflow *
        -   /
            ioba  literally '0003h',     /* iobyte address */
            sectorlen literally '128',   /* sector length */
            memsize address at(bdosa),   /* end of memory */
            rrec address at(rreca),      /* random record address */
            rovf byte at(rreco),         /* overflow on getfile */
            doll byte at(dolla),         /* dollar parameter */
            parm byte at(parma),         /* parameter */
            sizeset byte,                /* true if displaying size field
        -   */
            dpba address,                /* disk parameter block address *
        -   /
            dpb based dpba structure
            (spt address, bls byte, bms byte, exm byte, mxa address,
             dmx address, dbl address, cks address, ofs address),
            scptrk literally 'dpb.spt',
            blkshf literally 'dpb.bls',
            blkmsk literally 'dpb.bms',
            extmsk literally 'dpb.exm',
            maxall literally 'dpb.mxa',
            dirmax literally 'dpb.dmx',
            dirblk literally 'dpb.dbl',
            chksiz literally 'dpb.cks',
            offset literally 'dpb.ofs';

 5  1      boot: procedure external;
            /* reboot */
 6  2          end boot;

 7  1      mon1: procedure(f,a) external;
 8  2          declare f byte, a address;
 9  2          end mon1;

10  1      mon2: procedure(f,a) byte external;
11  2          declare f byte, a address;
12  2          end mon2;

13  1      mon3: procedure(f,a) address external;
14  2          declare f byte, a address;
```

```
15  2       end mon3;


16  1   status: procedure;
17  2       declare copyright(*) byte data (
            '   Copyright (c) 1979, Digital Research');
            /* dummy outer procedure 'status' will start at 100h */
            /* determine status of currently selected disk */

18  2   declare alloca address,
            /* alloca is the address of the disk allocation vector */
            alloc based alloca (1024) byte;  /* allocation vector */

19  2   declare
            true literally '1',
            false literally '0',
            forever literally 'while true',
            cr literally '13',
            lf literally '10';

20  2   printchar: procedure(char);
21  3       declare char byte;
22  3       call mon1(2,char);
23  3       end printchar;

24  2   crlf: procedure;
25  3       call printchar(cr);
26  3       call printchar(lf);
27  3       end crlf;

28  2   printb: procedure;
            /* print blank character */
29  3       call printchar(' ');
30  3       end printb;

31  2   printx: procedure(a);
32  3       declare a address;
33  3       declare s based a byte;
34  3           do while s <> 0;
35  4           call printchar(s);
36  4           a = a + 1;
37  4           end;
38  3       end printx;

39  2   print: procedure(a);
40  3       declare a address;
            /* print the string starting at address a until the
            next 0 is encountered */
41  3       call crlf;
42  3       call printx(a);
43  3       end print;

44  2   break: procedure byte;
45  3       return mon2(11,0); /* console ready */
46  3       end break;

47  2   declare dcnt byte;
```

```
 48   2      version: procedure byte;
                 /* returns current cp/m version # */
 49   3          return mon2(12,0);
 50   3          end version;

 51   2      select: procedure(d);
 52   3          declare d byte;
 53   3          call mon1(14,d);
 54   3          end select;

 55   2      open: procedure(fcb);
 56   3          declare fcb address;
 57   3          dcnt = mon2(15,fcb);
 58   3          end open;

 59   2      search: procedure(fcb);
 60   3          declare fcb address;
 61   3          dcnt = mon2(17,fcb);
 62   3          end search;

 63   2      searchn: procedure;
 64   3          dcnt = mon2(18,0);
 65   3          end searchn;

 66   2      cselect: procedure byte;
                 /* return current disk number */
 67   3          return mon2(25,0);
 68   3          end cselect;

 69   2      setdma: procedure(dma);
 70   3          declare dma address;
 71   3          call mon1(26,dma);
 72   3          end setdma;

 73   2      getalloca: procedure address;
                 /* get base address of alloc vector */
 74   3          return mon3(27,0);
 75   3          end getalloca;

 76   2      getlogin: procedure address;
                 /* get the login vector */
 77   3          return mon3(24,0);
 78   3          end getlogin;

 79   2      writeprot: procedure;
                 /* write protect the current disk */
 80   3          call mon1(28,0);
 81   3          end writeprot;

 82   2      getrodisk: procedure address;
                 /* get the read-only disk vector */
 83   3          return mon3(29,0);
 84   3          end getrodisk;

 85   2      setind: procedure;
                 /* set file indicators for current fcb */
```

```
 86   3          call mon1(30,fcba);
 87   3          end setind;

 88   2      set$dpb: procedure;
                 /* set disk parameter block values */
 89   3          dpba = mon3(31,0); /* base of dpb */
 90   3          end set$dpb;

 91   2      getuser: procedure byte;
                 /* return current user number */
 92   3          return mon2(32,0ffh);
 93   3          end getuser;

 94   2      setuser: procedure(user);
 95   3          declare user byte;
 96   3          call mon1(32,user);
 97   3          end setuser;

 98   2      getfilesize: procedure(fcb);
 99   3          declare fcb address;
100   3          call mon1(35,fcb);
101   3          end getfilesize;

102   2      declare oldsp address,   /* sp on entry */
                  stack(16) address;   /* this program's stack */

103   2      declare
                  fcbmax literally '512', /* max fcb count */
                  fcbs literally 'memory',/* remainder of memory */
                  fcb(33) byte at (fcba), /* default file control block */
                  buff(128) byte at (buffa),      /* default buffer */
                  ioval byte at (ioba);            /* io byte */

104   2      declare bpb address; /* bytes per block */

105   2      set$bpb: procedure;
106   3          call set$dpb; /* disk parameters set */
107   3          bpb = shl(double(1),blkshf) * sectorlen;
108   3          end set$bpb;

109   2      select$disk: procedure(d);
110   3          declare d byte;
                  /* select disk and set bpb */
111   3          call select(d);
112   3          call set$bpb; /* bytes per block */
113   3          end select$disk;

114   2      getalloc: procedure(i) byte;
                  /* return the ith bit of the alloc vector */
115   3          declare i address;
116   3          return
                  rol(alloc(shr(i,3)), (i and 111b) + 1);
117   3          end getalloc;

118   2      declare
                  accum(4) byte,     /* accumulator */
                  ibp byte;          /* input buffer pointer */
```

```
119   2       compare: procedure(a) byte;
                  /* compare accumulator with four bytes addressed by a */
120   3           declare a address;
121   3           declare (s based a) (4) byte;
122   3           declare i byte;
123   3               do i = 0 to 3;
124   4               if s(i) <> accum(i) then return false;
126   4               end;
127   3           return true;
128   3           end compare;

129   2       scan: procedure;
                  /* fill accum with next input value */
130   3           declare (i,b) byte;
131   3           setacc: procedure(b);
132   4               declare b byte;
133   4               accum(i) = b; i = i + 1;
135   4               end setacc;
                  /* deblank input */
136   3               do while buff(ibp) = ' '; ibp=ibp+1;
138   4               end;
                  /* initialize accum length */
139   3           i = 0;
140   3               do while i < 4;
141   4               if (b := buff(ibp)) > 1 then /* valid */
142   4                   call setacc(b); else /* blank fill */
143   4                   call setacc(' ');
144   4               if b <= 1 or b = ',' or b = ':' or
                      b = '*' or b = '.' or b = '>' or
                      b = '<' or b = '=' then buff(ibp) = 1;
                    else
146   4                   ibp = ibp + 1;
147   4               end;
148   3           ibp = ibp + 1;
149   3           end scan;

150   2       pdecimal: procedure(v,prec);
                  /* print value v with precision prec (10,100,1000)
                  with leading zero suppression */
151   3           declare
                      v address,     /* value to print */
                      prec address, /* precision */
                      zerosup byte, /* zero suppression flag */
                      d byte;       /* current decimal digit */
152   3           zerosup = true;
153   3               do while prec <> 0;
154   4               d = v / prec ; /* get next digit */
155   4               v = v mod prec;/* get remainder back to v */
156   4               prec = prec / 10; /* ready for next digit */
157   4               if prec <> 0 and zerosup and d = 0 then call printb; else
159   4                   do; zerosup = false; call printchar('0'+d);
162   5                   end;
163   4               end;
164   3           end pdecimal;

165   2       add$block: procedure(ak,ab);
```

```
166   3          declare (ak, ab) address;
                 /* add one block to the kilobyte accumulator */
167   3          declare kaccum based ak address; /* kilobyte accum */
168   3          declare baccum based ab address; /* byte accum */
169   3          baccum = baccum + bpb;
170   3              do while baccum >= 1024;
171   4              baccum = baccum - 1024;
172   4              kaccum = kaccum + 1;
173   4              end;
174   3          end add$block;

175   2      count: procedure(mode) address;
176   3          declare mode byte; /* true if counting 0's */
                 /* count kb remaining, kaccum set upon exit */
177   3          declare
                     ka  address,  /* kb accumulator */
                     ba  address,  /* byte accumulator */
                     i   address,  /* local index */
                     bit byte;     /* always 1 if mode = false */
178   3          ka, ba = 0;
179   3          bit = 0;
180   3              do i = 0 to maxall;
181   4              if mode then bit = getalloc(i);
183   4              if not bit then call add$block(.ka,.ba);
185   4              end;
186   3          return ka;
187   3          end count;

188   2      abortmsg: procedure;
189   3          call print(.('** Aborted **',0));
190   3          end abortmsg;

191   2      userstatus: procedure;
                 /* display active user numbers */
192   3          declare i byte;
193   3          declare user(32) byte;
194   3          declare ufcb(*) byte data ('???????????',0,0,0);
195   3          call print(.('Active User :',0));
196   3          call pdecimal(getuser,10);
197   3          call print(.('Active Files:',0));
198   3              do i = 0 to last(user);
199   4              user(i) = false;
200   4              end;
201   3          call setdma(.fcbs);
202   3          call search(.ufcb);
203   3              do while dcnt <> 255;
204   4              if (i := fcbs(shl(dcnt and 11b,5))) <> 0e5h then
205   4                  user(i and 1fh) = true;
206   4              call searchn;
207   4              end;
208   3              do i = 0 to last(user);
209   4              if user(i) then call pdecimal(i,10);
211   4              end;
212   3          end userstatus;

213   2      drivestatus: procedure;
214   3          declare
```

```
                    rpb address,
                    rpd address;
215   3           pv: procedure(v);
216   4               declare v address;
217   4               call crlf;
218   4               call pdecimal(v,10000);
219   4               call printchar(':');
220   4               call printb:
221   4               end pv;
                  /* print the characteristics of the currently selected drive *
          -   /
222   3           call print(.('    ',0));
223   3           call printchar(cselect+'A');
224   3           call printchar(':');
225   3           call printx(.(' Drive Characteristics',0));
226   3           rpb = shl(double(1),blkshf); /* records/block=2**blkshf */
227   3           if (rpd := (maxall+1) * rpb) = 0 and (rpb <> 0) then
228   3               call print(.('65536: ',0)); else
229   3               call pv(rpd);
230   3               call printx(.('128 Byte Record Capacity',0));
231   3           call pv(count(false));
232   3               call printx(.('Kilobyte Drive  Capacity',0));
233   3           call pv(dirmax+1);
234   3               call printx(.('32  Byte Directory Entries',0));
235   3           call pv(shl(chksiz,2));
236   3               call printx(.('Checked  Directory Entries',0));
237   3           call pv((extmsk+1) * 128);
238   3               call printx(.('Records/ Extent',0));
239   3           call pv(rpb);
240   3               call printx(.('Records/ Block',0));
241   3           call pv(scptrk);
242   3               call printx(.('Sectors/ Track',0));
243   3           call pv(offset);
244   3               call printx(.('Reserved Tracks',0));
245   3           call crlf;
246   3           end drivestatus;

247   2       diskstatus: procedure;
                  /* display disk status */
248   3           declare login address, d byte;
249   3           login = getlogin; /* login vector set */
250   3           d = 0;
251   3               do while login <> 0;
252   4               if low(login) then
253   4                   do; call select$disk(d);
255   5                   call drivestatus;
256   5                   end;
257   4               login = shr(login,1);
258   4               d = d + 1;
259   4               end;
260   3           end diskstatus;

261   2       match: procedure(va,vl) byte;
                  /* return index+1 to vector at va if match */
262   3           declare va address,
                        v based va (16) byte,
                        vl byte;
```

```
263   3          declare (i,j,match,sync) byte;
264   3          j,sync = 0;
265   3              do sync = 1 to vl;
266   4              match = true;
267   4                  do i = 0 to 3;
268   5                  if v(j) <> accum(i) then match=false;
270   5                  j = j + 1;
271   5                  end;
272   4              if match then return sync;
274   4              end;
275   3          return 0; /* no match */
276   3          end match;

277   2      declare devl(*) byte data
                   ('CON:RDR:PUN:LST:DEV:VAL:USR:DSK:');

278   2      devreq: procedure byte;
                   /* process device request, return true if found */
                   /* device tables */
279   3          declare
                       devr(*) byte data
                       (/* console */ 'TTY:CRT:BAT:UC1:',
                        /* reader  */ 'TTY:PTR:UR1:UR2:',
                        /* punch   */ 'TTY:PTP:UP1:UP2:',
                        /* listing */ 'TTY:CRT:LPT:UL1:');

280   3          declare
                       (i,j,iobyte,items) byte;


281   3          prname: procedure(a);
282   4              declare a address,
                           x based a byte;
                       /* print device name at a */
283   4                  do while x <> ':';
284   5                  call printchar(x); a=a+1;
286   5                  end;
287   4              call printchar(':');
288   4              end prname;

289   3          items = 0;
290   3              do forever;
291   4              call scan;
292   4              if (i:=match(.devl,8)) = 0 then return items<>0;
294   4              items = items+1; /* found first/next item */
295   4              if i = 5 then /* device status request */
296   4                  do;
297   5                  iobyte = ioval; j = 0;
299   5                      do i = 0 to 3;
300   6                      call prname(.devl(shl(i,2)));
301   6                      call printx(.(' is ',0));
302   6                      call prname(.devr(shl(iobyte and 11b,2)+j));
303   6                      j = j + 16; iobyte = shr(iobyte,2);
305   6                      call crlf;
306   6                      end;
307   5                  end; else /* not dev: */
308   4              if i = 6 then /* list possible assignment */
```

```
 309   4                  do;
 310   5                  call print(.('Temp R/O Disk: d:=R/O',0));
 311   5                  call print(.('Set Indicator: d:filename.typ ',
                                    '$R/O $R/W $SYS $DIR',0));
 312   5                  call print(.('Disk Status  : DSK: d:DSK:',0));
 313   5                  call print(.('User Status  : USR:',0));
 314   5                  call print(.('Iobyte Assign:',0));
 315   5                     do i = 0 to 3; /* each line shows one device */
 316   6                     call crlf;
 317   6                     call prname(.devl(shl(i,2)));
 318   6                     call printx(.(' =',0));
 319   6                        do j = 0 to 12 by 4;
 320   7                        call printchar(' ');
 321   7                        call prname(.devr(shl(i,4)+j));
 322   7                        end;
 323   6                     end;
 324   5                  end; else
 325   4               if i = 7 then /* list user status values */
 326   4                  do; call userstatus;
 328   5                  return true;
 329   5                  end; else
 330   4            if i = 8 then /* show the disk device status */
 331   4                  call diskstatus; else
                     /* scan item i-1 in device table */
 332   4                  do; /* find base of destination */
 333   5                  j = shl(i:=i-1,4);
 334   5                  call scan;
 335   5                  if accum(0) <> '=' then
 336   5                     do; call print(.('Bad Delimiter',0));
 338   6                     return true;
 339   6                     end;
 340   5                  call scan;
 341   5                  if (j:=match(.devr(j),4)-1) = 255 then
 342   5                     do; call print(.('Invalid Assignment',0));
 344   6                     return true;
 345   6                     end;
 346   5                  iobyte = 1111$1100b; /* construct mask */
 347   5                     do while (i:=i-1) <> 255;
 348   6                     iobyte = rol(iobyte,2);
 349   6                     j = shl(j,2);
 350   6                     end;
 351   5                  ioval = (ioval and iobyte) or j;
 352   5                  end;
                     /* end of current item, look for more */
 353   4                  call scan;
 354   4                  if accum(0) = ' ' then return true;
 356   4                  if accum(0) <> ',' then
 357   4                     do; call print(.('Bad Delimiter',0));
 359   5                     return true;
 360   5                     end;
 361   4                  end; /* of do forever */
 362   3            end devreq;

 363   2         pvalue: procedure(v);
 364   3            declare (d,zero) byte,
                        (k,v) address;
 365   3            k = 10000;
```

```
366   3          zero = false;
367   3              do while k <> 0;
368   4              d = low(v/k); v = v mod k;
370   4              k = k / 10;
371   4              if zero or k = 0 or d <> 0 then
372   4                   do; zero = true; call printchar('0'+d);
375   5                   end;
376   4              end;
377   3          call printchar('k');
378   3          call crlf;
379   3          end pvalue;

380   2      comp$alloc: procedure;
381   3          alloca = getalloca;
382   3          call printchar(cselect+'A');
383   3          call printx(.(': ',0));
384   3          end comp$alloc;

385   2      prcount: procedure;
                  /* print the actual byte count */
386   3          call pvalue(count(true));
387   3          end prcount;

388   2      pralloc: procedure;
                  /* print allocation for current disk */
389   3          call print (.('Bytes Remaining On ',0));
390   3          call comp$alloc;
391   3          call prcount;
392   3          end pralloc;

393   2      prstatus: procedure;
                  /* print the status of the disk system */
394   3          declare (login, rodisk) address;
395   3          declare d byte;
396   3          login = getlogin; /* login vector set */
397   3          rodisk = getrodisk; /* read only disk vector set */
398   3          d = 0;
399   3              do while login <> 0;
400   4              if low(login) then
401   4                  do; call select$disk(d);
403   5                  call comp$alloc;
404   5                  call printx(.('R/',0));
405   5                  if low(rodisk) then
406   5                      call printchar('0'); else
407   5                      call printchar('W');
408   5                  call printx(.(', Space: ',0));
409   5                  call prcount;
410   5                  end;
411   4              login = shr(login,1); rodisk = shr(rodisk,1);
413   4              d = d + 1;
414   4              end;
415   3          call crlf;
416   3          end prstatus;

417   2      setdisk: procedure;
418   3          if fcb(0) <> 0 then call select$disk(fcb(0)-1);
420   3          end setdisk;
```

```
421   2       getfile: procedure;
                  /* process file request */

422   3           declare
                      fnam literally '11',   fext literally '12',
                      fmod literally '14',
                      frc  literally '15',   fln  literally '15',
                      fdm  liter lly '16',   fdl  literally '31',
                      ftyp literally '9',
                      rofile literally '9', /* read/only file */
                      infile literally '10';  /* invisible file */
423   3           declare
                      fcbn address,   /* number of fcb's collected so far */
                      finx(fcbmax) address, /* index vector used during sort */
                      fcbe(fcbmax) address, /* extent counts */
                      fcbb(fcbmax) address, /* byte count (mod kb) */
                      fcbk(fcbmax) address, /* kilobyte count */
                      fcbr(fcbmax) address, /* record count */
                      bfcba address,    /* index into directory buffer */
                      fcbsa address,    /* index into fcbs */
                      bfcb based bfcba (32) byte,   /* template over directory *
          -   /
                      fcbv based fcbsa (16) byte;   /* template over fcbs entry
          -   */
424   3           declare
                      i address,     /* fcb counter during collection and displa
          -   y */
                      l address,     /* used during sort and display */
                      k address,     /* " */
                      m address,     /* " */
                      kb byte,       /* byte counter */
                      lb byte,       /* byte counter */
                      mb byte,       /* byte counter */
                      (b,f) byte,    /* counters */
                      matched byte;  /* used during fcbs search */

425   3           multi16: procedure;
                      /* utility to compute fcbs address from i */
426   4               fcbsa = shl(i,4) + .fcbs;
427   4               end multi16;

428   3           declare
                      scase byte; /* status case # */

429   3           declare
                      fstatlist(*) byte data('R/O',0,'R/W',0,'SYS',0,'DIR',0);

430   3           setfilestatus: procedure byte;
                          /* eventually, scase set r/o=0,r/w=1,dat=2,sys=3 */
431   4               declare
                          fstat(*) byte data('R/O R/W SYS DIR ');
432   4               if doll = ' ' then return false;
434   4               call move(4,.parm,.accum); /* $???? */
435   4               if accum(0) = 'S' and accum(1) = ' ' then
436   4                   return not (sizeset := true);
                      /* must be a parameter */
```

```
437  4              if (scase := match(.fstat,4)) = 0 then
438  4                  call print(.('Invalid File Indicator',0));
439  4              return true;
440  4              end setfilestatus;

441  3          printfn: procedure;
442  4              declare (k, lb) byte;
                    /* print file name */
443  4                  do k = 1 to fnam;
444  5                  if (lb := fcbv(k) and 7fh) <> ' ' then
445  5                      do; if k = ftyp then call printchar('.');
448  6                      call printchar(lb);
449  6                      end;
450  5                  end;
451  4              end printfn;

452  3          call set$bpb; /* in case default disk */
453  3          call setdisk;
454  3          sizeset = false;
455  3          scase = 255;
456  3          if setfilestatus then
457  3              do; if scase = 0 then return;
460  4              scase = scase - 1;
461  4              end; else
462  3          if fcb(1) = ' ' then /* no file named */
463  3              do; call pralloc;
465  4              return;
466  4              end;
                   /* read the directory, collect all common file names */
                   fcbn,fcb(0) = 0;
468  3          fcb(fext),fcb(fmod) = '?'; /* question mark matches all */
469  3          call search(fcba); /* fill directory buffer */
470  3          collect: /* label for debug */
                    do while dcnt <> 255;
                    /* another item found, compare it for common entry */
471  4              bfcba = shl(dcnt and 11b,5)+buffa; /* dcnt mod 4 * 32 */
472  4              matched = false; i = 0;
474  4                  do while not matched and i < fcbn;
                        /* compare current entry */
475  5                  call multi16;
476  5                      do kb = 1 to fnam;
477  6                      if bfcb(kb) <> fcbv(kb) then kb = fnam; else
                            /* complete match if at end */
479  6                          matched = kb = fnam;
480  6                      end;
481  5                  i = i + 1;
482  5                  end;
483  4          checkmatched: /* label for debug */
                    if matched then i = i - 1; else
485  4                  do; /* copy to new position in fcbs */
486  5                  fcbn = (i := fcbn) + 1;
487  5                  call multi16;
                        /* fcbsa set to next to fill */
488  5                  if (fcbn > fcbmax) or (fcbsa + 16) >= memsize then
489  5                      do; call print(.('** Too Many Files **',0));
491  6                      i = 0; fcbn = 1;
493  6                      call multi16;
```

```
494   6                     end;
                         /* save index to element for later sort */
495   5                     finx(i) = i;
496   5                     do kb = 0 to fnam;
497   6                     fcbv(kb) = bfcb(kb);
498   6                     end;
499   5                     fcbe(i),fcbb(i),fcbk(i),fcbr(i) = 0;
500   5                     end;
                    /* entry is at, or was placed at location i in fcbs */
501   4                 fcbe(i) = fcbe(i) + 1; /* extent incremented */
                        /* record count */
502   4                 fcbr(i) = fcbr(i) + bfcb(frc)
                              + (bfcb(fext) and extmsk) * 128;
                        /* count kilobytes */
503   4               countbytes: /* label for debug */
                        lb = 1;
504   4                 if maxall > 255 then lb = 2; /* double precision inx */
506   4                     do kb = fdm to fdl by lb;
507   5                     mb = bfcb(kb);
508   5                     if lb = 2 then /* double precision inx */
509   5                         mb = mb or bfcb(kb+1);
510   5                     if mb <> 0 then /* allocated */
511   5                     call add$block(.fcbk(i),.fcbb(i));
512   5                     end;
513   4                 call searchn; /* to next entry in directory */
514   4                 end; /* of do while dcnt <> 255 */

515   3               display: /* label for debug */
                        /* now display the collected data */
                        if fcbn = 0 then call print(.('File Not Found',0)); else
517   3                 if scase = 255 then /* display collected data */
518   3                     do;
                            /* sort the file names in ascending order */
519   4                     if fcbn > 1 then /* requires at least two to sort */
520   4                         do; l = 1;
522   5                         do while l > 0; /* bubble sort */
523   6                         l = 0;
524   6                             do m = 0 to fcbn - 2;
525   7                             i = finx(m+1); call multi16; bfcba = fcbsa; i
      -   = finx(m);
529   7                             call multi16; /* sets fcbsa, basing fcbv */
530   7                                 do kb = 1 to fnam; /* compare for less or
      -   equal */
531   8                             if (b:=bfcb(kb)) < (f:=fcbv(kb)) then /* s
      -   witch */
532   8                                 do; k = finx(m); finx(m) = finx(m + 1)
      -   ;
535   9                                 finx(m + 1) = k; l = l + 1; kb = fnam;
538   9                                 end;
539   8                             else if b > f then kb = fnam;
      -   are */
                                    end;
542   7                         end;
543   6                     end;
544   5                 end;
545   4                 if sizeset then
546   4                     call print(.(' Size ',0)); else
```

/* stop comp

```
547   4                 call crlf;
548   4               call printx(.(' Recs  Bytes  Ext Acc',0));
549   4               l = 0;
550   4                   do while l < fcbn;
551   5                   i = finx(l); /* i is the index to next in order */
552   5                   call multi16; call crlf;
                          /* print the file length */
554   5                   call move(16,.fcbv(0),fcba);
555   5                   fcb( ) = 0;
556   5                   if sizeset then
557   5                       do; call getfilesize(fcba);
559   6                       if rovf <> 0 then call printx(.('65536',0)); else
561   6                           call pdecimal(rrec,10000);
562   6                       call printb;
563   6                       call printb;
564   5                   call pdecimal(fcbr(i),10000); /* rrrrr */
565   5                   call printb; /* blank */
566   5                   call pdecimal(fcbk(i),10000);  /* bbbbbk */
567   5                   call printchar('k'); call printb;
569   5                   call pdecimal(fcbe(i),1000);   /* eeee */
570   5                   call printb;
571   5                   call printchar('R');
572   5                   call printchar('/');
573   5                   if rol(fcbv(rofile),1) then
574   5                       call printchar('0'); else
575   5                       call printchar('W');
576   5                   call printb;
577   5                   call printchar('A'+cselect); call printchar(':');
                          /* print filename.typ */
579   5                   if (mb:=rol(fcbv(infile),1)) then call printchar('(');
581   5                   call printfn;
582   5                   if mb then call printchar(')');
584   5                   l = l + 1;
585   5                   end;
586   4               call pralloc;
587   4               end; else
588   3           setfileatt: /* label for debug */
                  /* set file attributes */
                      do;
589   4                   l = 0;
590   4                       do while l < fcbn;
591   5                       if break then
592   5                           do; call abortmsg; return;
595   6                           end;
596   5                       i = l;
597   5                       call multi16;
598   5                       call crlf;
599   5                       call printfn;
600   5                           do case scase;
                                  /* set to r/o */
601   6                           fcbv(rofile) = fcbv(rofile) or 80h;
                                  /* set to r/w */
602   6                           fcbv(rofile) = fcbv(rofile) and 7fh;
                                  /* set to sys */
603   6                           fcbv(infile) = fcbv(infile) or 80h;
                                  /* set to dir */
604   6                           fcbv(infile) = fcbv(infile) and 7fh;
```

```
605   6                   end;
                        /* place name into default fcb location */
606   5                 call move(16,fcbsa,fcba);
607   5                 fcb(0) = 0; /* in case matched user# > 0 */
608   5                 call setind; /* indicators set */
609   5                 call printx(.(' set to ',0));
610   5                 call printx(.fstatlist(shl(scase,2)));
611   5                 l = l + 1;
612   5                 end;
613   4             end;
614   3         end getfile;

615   2     setdrivestatus: procedure;
            /* handle possible drive status assignment */
616   3         call scan; /* remove drive n me */
617   3         call scan; /* check for = */
618   3         if accum(0) = '=' then
619   3             do; call scan; /* get assignment */
621   4             if compare(.('R/O ')) then
622   4                 do; call setdisk; /* a: ... */
624   5                 call writeprot;
625   5                 end; else
626   4             call print(.('Invalid Disk Assignment',0));
627   4             end;
            else /* not a disk assignment */
628   3             do; call setdisk;
630   4             if match(.dev1,8) = 8 then call drive$status; else
632   4                 call getfile;
633   4             end;
634   3         end setdrivestatus;

           /* save stack pointer and reset */
635   2     oldsp = stackptr;
636   2     stackptr = .stack(length(stack));
           /* process request */
637   2     if version < cpmversion then
638   2         call print(.('Wrong CP/M Version (Requires 2.0)',0));
              else
639   2             do;
                    /* size display if $S set in command */
640   3             ibp = 1; /* initialize buffer pointer */
641   3             if fcb(0) = 0 and fcb(1) = ' ' then /* stat only */
642   3                 call prstatus; else
643   3                 do;
644   4                 if fcb(0) <> 0 then
645   4                     call setdrivestatus; else
646   4                     do;
647   5                     if not devreq then /* must be file name */
648   5                         call getfile;
649   5                     end;
650   4                 end;
651   3             end;
           /* restore old stack before exit */
652   2         stackptr = oldsp;
653   2         end status;
654   1     end;
           EOF
```

0000 STAT#
0000 STAT#

0433 16	0490 20	0494 22	049F 23	04A0 24
04A0 25	04A5 26	04AA 27	04AB 28	04AB 29
04B0 30	04B1 31	04B7 34	04C0 35	04C7 36
04CE 37	04D1 38	04D2 39	04D8 41	04DB 42
04E3 43	04E4 44	04E4 45	04ED 46	04ED 48
04ED 49	04F6 50	04F6 51	04FA 53	0505 54
0506 55	050C 57	0518 58	0519 59	051F 61
052B 62	052C 63	052C 64	0537 65	0538 66
0538 67	0541 68	0541 69	0547 71	0550 72
0551 73	0551 74	055A 75	055A 76	055A 77
0563 78	0563 79	0563 80	056B 81	056C 82
056C 83	0575 84	0575 85	0575 86	057D 87
057E 88	057E 89	0589 90	058A 91	058A 92
0593 93	0593 94	0597 96	05A2 97	05A3 98
05A9 100	05B2 101	05B3 105	05B3 106	05B6 107
05CB 108	05CC 109	05D0 111	05D7 112	05DA 113
05DB 114	05E1 116	05FE 117	05FE 119	0604 123
0612 124	062C 125	062F 126	0636 127	0639 128
0639 129	06EB 131	06EF 133	06FC 134	0700 135
0639 136	0648 137	064C 138	064F 139	0654 140
065C 141	0670 142	067A 143	067F 144	06D1 145
06DF 146	06E3 147	06E6 148	06EA 149	0701 150
070B 152	0710 153	071C 154	072A 155	0734 156
0740 157	075B 158	0761 160	0766 161	076F 162
076F 163	0772 164	0773 165	077D 169	078C 170
0799 171	07A8 172	07B3 173	07B6 174	07B7 175
07BB 178	07C4 179	07C8 180	07DF 181	07E6 182
07F1 183	07F8 184	0801 185	080E 186	0812 187
0812 188	0812 189	0818 190	0819 191	0819 195
081F 196	082B 197	0831 198	083F 199	084A 200
0851 201	0857 202	085D 203	0865 204	087F 205
088D 206	0890 207	0893 208	08A1 209	08AF 210
08BB 211	08C2 212	08C3 213	09C0 215	09C6 217
09C9 218	09D4 219	09D9 220	09DC 221	08C3 222
08C9 223	08D2 224	08D7 225	08DD 226	08EC 227
091C 228	0925 229	092D 230	0933 231	093D 232
0943 233	0951 234	0957 235	0969 236	096F 237
0986 238	098C 239	0994 240	099A 241	09A3 242
09A9 243	09B6 244	09BC 245	09BF 246	09DD 247
09DD 249	09E3 250	09E8 251	09F4 252	09FC 254
0A03 255	0A06 256	0A06 257	0A13 258	0A15 259
0A18 260	0A19 261	0A21 264	0A2B 265	0A37 266
0A3C 267	0A4A 268	0A64 269	0A69 270	0A6D 271
0A72 272	0A79 273	0A7D 274	0A84 275	0A87 276
0A87 278	0C69 281	0C6F 283	0C78 284	0C7F 285
0C86 286	0C89 287	0C8E 288	0A87 289	0A8C 290
0A8C 291	0A8F 292	0A9F 293	0AA8 294	0AAC 295
0AB4 297	0ABA 298	0ABF 299	0ACB 300	0ADC 301
0AE2 302	0AF9 303	0B01 304	0B0B 305	0B0E 306
0B15 307	0B18 308	0B20 310	0B26 311	0B2C 312
0B32 313	0B38 314	0B3E 315	0B4C 316	0B4F 317
0B60 318	0B66 319	0B85 320	0B8A 321	0B8C 322
0BA4 323	0BAB 324	0BAE 325	0BB6 327	0BB9 328

0BBC 329	0BBF 330	0BC7 331	0BCD 333	0BDB 334
0BDE 335	0BE6 337	0BEC 338	0BEF 339	0BEF 340
0BF2 341	0C0B 343	0C11 344	0C14 345	0C14 346
0C19 347	0C25 348	0C2D 349	0C35 350	0C38 351
0C46 352	0C46 353	0C49 354	0C51 355	0C54 356
0C5C 358	0C62 359	0C65 360	0C65 361	0C68 362
0C8F 363	0C95 365	0C9B 366	0CA0 367	0CAC 368
0CBA 369	0CC4 370	0CD0 371	0CEB 373	0CF0 374
0CF9 375	0CF9 376	0CFC 377	0D01 378	0D04 379
0D05 380	0D05 381	0D0B 382	0D14 383	0D1A 384
0D1B 385	0D1B 386	0D25 387	0D26 388	0D26 389
0D2C 390	0D2F 391	0D32 392	0D33 393	0D33 396
0D39 397	0D3F 398	0D44 399	0D50 400	0D58 402
0D5F 403	0D62 404	0D68 405	0D70 406	0D78 407
0D7D 408	0D83 409	0D86 410	0D86 411	0D93 412
0DA0 413	0DA2 414	0DA5 415	0DA8 416	0DA9 417
0DA9 418	0DB1 419	0DB9 420	0DBA 421	135D 425
135D 426	136B 427	136C 430	136C 432	1374 433
1377 434	1387 435	139F 436	13A7 437	13B7 438
13BD 439	13C0 440	13C0 441	13C0 443	13CE 444
13E3 446	13EB 447	13F0 448	13F7 449	13F7 450
1401 451	0DBA 452	0DBD 453	0DC0 454	0DC5 455
0DCA 456	0DD1 458	0DD9 459	0DDA 460	0DE1 461
0DE4 462	0DEC 464	0DEF 465	0DF0 466	0DF0 467
0DFA 468	0E04 469	0E0A 470	0E12 471	0E24 472
0E29 473	0E2F 474	0E45 475	0E48 476	0E56 477
0E71 478	0E79 479	0E84 480	0E8E 481	0E95 482
0E98 483	0E9F 484	0EA9 486	0EB3 487	0EB6 488
0ED8 490	0EDE 491	0EE4 492	0EEA 493	0EED 494
0EED 495	0EFE 496	0F0C 497	0F24 498	0F2E 499
0F60 500	0F60 501	0F79 502	0FB9 503	0FBE 504
0FCD 505	0FD2 506	0FF1 507	0FFF 508	1007 509
101C 510	1024 511	103A 512	103D 513	1040 514
1043 515	104F 516	1058 517	1060 519	106B 521
1071 522	107C 523	1082 524	1097 525	10A6 526
10A9 527	10AF 528	10BE 529	10C1 530	10CF 531
10F3 533	1102 534	111B 535	112C 536	1133 537
1138 538	113B 539	1145 540	114A 541	1154 542
1161 543	1164 544	1164 545	116B 546	1174 547
1177 548	117D 549	1183 550	118F 551	119E 552
11A1 553	11A4 554	11B8 555	11BD 556	11C4 558
11CA 559	11D2 560	11DB 561	11E6 562	11E9 563
11E9 564	11FA 565	11FD 566	120E 567	1213 568
1216 569	1227 570	122A 571	122F 572	1234 573
1241 574	1249 575	124E 576	1251 577	125A 578
125F 579	126F 580	1274 581	1277 582	127E 583
1283 584	128A 585	128D 586	1290 587	1293 588
1293 589	1299 590	12A5 591	12AC 593	12AF 594
12B0 595	12B0 596	12B6 597	12B9 598	12BC 599
12BF 600	12CF 601	12E1 602	12F3 603	1305 604
1317 605	131F 606	1333 607	1338 608	133B 609
1341 610	1352 611	1359 612	135C 613	135C 614
1402 615	1402 616	1405 617	1408 618	1410 620
1413 621	141D 623	1420 624	1423 625	1426 626
142C 627	142F 629	1432 630	143F 631	1445 632

```
1448 633    1448 634    0433 635    043A 636    043E 637
0446 638    044F 640    0454 641    046C 642    0472 644
047A 645    0480 647    0488 648    048B 649    048B 650
048B 651    048B 652    048F 653
0000 MODULE#
```


SERIAL # _____