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


   1          PIPMOD:
              DO;
              /* P E R I P H E R A L  I N T E R C H A N G E  P R O G R A M  CP/M v.2.2

                    COPYRIGHT (C) 1976, 1977, 1978, 1979, 1980
                    DIGITAL RESEARCH
                    BOX 579
                    PACIFIC GROVE, CA
                    93950
                 */

   2   1      DECLARE
                 CPMVERSION LITERALLY '0020H'; /* REQUIRED FOR OPERATION */

   3   1      DECLARE
                 IOBYTE   BYTE EXTERNAL,     /* IOBYTE AT 0003H */
                 MAXB  ADDRESS EXTERNAL,     /* ADDR FIELD OF JMP BDOS */
                 FCB (33) BYTE EXTERNAL,     /* DEFAULT FILE CONTROL BLOCK */
                 BUFF(128)BYTE EXTERNAL;     /* DEFAULT BUFFER */

   4   1      DECLARE
                 ENDFILE LITERALLY '1AH',    /* END OF FILE MARK */
                 JMP   LITERALLY '0C3H',     /* 8080 JUMP INSTRUCTION */
                 RET   LITERALLY '0C9H';     /* 8080 RETURN */

              /* THE FIRST PORTION OF THE PIP PROGRAM 'FAKES' THE PAGE ONE
              (100H - 1FFH) SECTION OF PIP WHICH CONTAINS A JUMP TO PIPENTRY, AN
          -    D
              SPACE FOR CUSTOM I/O DRIVERS (WHICH CAN BE 'PATCHED' USING DDT) IN
          -     THE
              REMAINING PAGE ONE AREA.  THE PIP PROGRAM ACTUALLY STARTS AT 200H
          -    */

   5   1      DECLARE JUMP BYTE DATA(JMP); /* JMP INSTRUCTION TO */
              /* JMP .PIPENTRY-3 WHERE THE LXI SP,STACK ACTUALLY OCCURS */
   6   1      DECLARE JADR ADDRESS DATA(.PIPENTRY-3); /* START OF PIP */
   7   1      DECLARE INPSUB(3) BYTE DATA(RET,0,0);     /* INP: RET NOP NOP */
   8   1      DECLARE OUTSUB(3) BYTE DATA(RET,0,0);     /* OUT: RET NOP NOP */
   9   1      DECLARE INPDATA BYTE DATA(ENDFILE); /* RETURNED DATA */
                 /* NOTE:  PAGE 1 AT 100H CONTAINS THE FOLLOWING
                    100H:  JMP PIPENTRY    ;TO START THE PIP PROGRAM
                    103H:  RET             ;INP: DEFAULTS TO EMPTY INPUT (DATA 1AH
          -       AT 109H)
                    104H:  NOP
                    105H:  NOP
                    106H:  RET             ;OUT: DEFAULTS TO EMPTY OUTPUT
                    107H:  NOP
                    108H:  NOP
                    109H: 1AH=ENDFILE      ;DATA FROM INP: FUNCTION IS STORED HERE
          -       ON
                                           ;RETURN FROM THE INP: ENTRY POINT
```

```
                10AH: - 1FFH            ;SPACE RESERVED FOR SPECIAL PURPOSE
                ; DRIVERS - IF INCLUDED, THEN REPLACE 103H AND 106H BY JMP'S
                ; TO THE PROPER LOCATIONS WITHIN THE RESERVED AREA.
                ;  ALSO, RETURN DATA FROM INP: ENTRY POINT AT 109H.
                ; THESE DRIVERS ARE MOST EASILY INSERTED WITH THE DDT PROGRAM
                ; UNDER CP/M
                */

10    1       DECLARE /* 16 BYTE MESSAGE */
                  FREEMEMORY LITERALLY '''(INP:/OUT:SPACE)''',
                  /* 256 BYTE AREA FOR INP: OUT: PATCHING */
                  RESERVED(*) BYTE DATA(0,0,0,0,0,0,
                  FREEMEMORY, FREEMEMORY, FREEMEMORY,
                  FREEMEMORY, FREEMEMORY, FREEMEMORY, FREEMEMORY,
                  FREEMEMORY, FREEMEMORY, FREEMEMORY, FREEMEMORY,
                  FREEMEMORY, FREEMEMORY, FREEMEMORY, FREEMEMORY);



11    1           DECLARE COPYRIGHT(*) BYTE DATA (
                  '   COPYRIGHT (C) 1979, DIGITAL RESEARCH,  PIP VERS 1.5');

12    1           DECLARE INPLOC ADDRESS DATA (.INPSUB);  /* ADDRESS OF INP: DEV
          -    ICE */
13    1           DECLARE OUTLOC ADDRESS DATA (.OUTSUB);  /* ADDRESS OF OUT: DEV
          -    ICE */

14    1       OUT: PROCEDURE(B);
15    2           DECLARE B BYTE;
                  /* SEND B TO OUT: DEVICE */
16    2           CALL OUTLOC;
17    2           END OUT;

18    1       INP: PROCEDURE BYTE;
19    2           CALL INPLOC;
20    2           RETURN INPDATA;
21    2           END INP;


22    1       TIMEOUT: PROCEDURE;
                  /* WAIT FOR 50 MSEC */
23    2           CALL TIME(250); CALL TIME(250);
25    2           END TIMEOUT;

                /* LITERAL DECLARATIONS */
26    1         DECLARE
                    LIT LITERALLY 'LITERALLY',
                    LPP LIT '60',     /* LINES PER PAGE */
                    TAB LIT '09H',    /* HORIZONTAL TAB */
                    FF  LIT '0CH',    /* FORM FEED */
                    LA  LIT '05FH',   /* LEFT ARROW */
                    LB  LIT '05BH',   /* LEFT BRACKET */
                    RB   LIT '05DH',  /* RIGHT BRACKET */
                    XOFF LIT '13H', /* TRANSMIT BUFFER FUNCTION */

                    RDR LIT '5',
```

```
            LST LIT '10',
            PUNP LIT '15',                  /* POSITION OF 'PUN' + 1 */
            CONP LIT '19',                  /* CONSOLE */
            NULP LIT '19',                  /* NUL: BEFORE INCREMENT */
            EOFP LIT '20',                  /* EOF: BEFORE INCREMENT */
            HSRDR LIT 'RDR',                /* READER DEVICES */
            PRNT  LIT '10',                 /* PRINTER */

            FSIZE LIT '33',
            FRSIZE LIT '36', /* SIZE OF RANDOM FCB */
            NSIZE LIT '8',
            FNSIZE LIT '11',
            MDISK LIT '1',
            FNAM LIT '8',
            FEXT LIT '9',
            FEXTL LIT '3',
            ROFILE  LITERALLY '9',  /* READ ONLY FILE FIELD */
            SYSFILE LITERALLY '10', /* SYSTEM FILE FIELD */
            FREEL LIT '12',  /* REEL NUMBER FIELD OF FCB */

            HBUFS LIT '80',                 /* "HEX" BUFFER SIZE */

            ERR LIT '0',
            SPECL LIT '1',
            FILE LIT '2',
            PERIPH LIT '3',
            DISKNAME LIT '4';

27  1       DECLARE
            COLUMN BYTE,     /* COLUMN COUNT FOR PRINTER TABS */
            LINENO BYTE,     /* LINE WITHIN PAGE */
            AMBIG BYTE,                     /* SET FOR AMBIGUOUS FILE REFS *
          -   /
            PARSET BYTE,                    /* TRUE IF PARAMETERS PRESENT */
            FEEDBASE BYTE,                  /* USED TO FEED SEARCH CHARACTER
          -   S */
            FEEDLEN BYTE,                   /* LENGTH OF FEED STRING */
            MATCHLEN BYTE,                  /* USED IN MATCHING STRINGS */
            QUITLEN BYTE,                   /* USED TO TERMINATE QUIT COMMAN
          -   D */
            NBUF BYTE,                      /* NUM BUFFERS-1 IN SBUFF AND DB
          -   UFF */
            CDISK BYTE,                     /* CURRENT DISK */
            BUFFER LITERALLY 'BUFF',        /* DEFAULT BUFFER */
            SEARFCB LITERALLY 'FCB',        /* SEARCH FCB IN MULTI COPY */
            MEMSIZE LITERALLY 'MAXB',       /* MEMORY SIZE */
            SBLEN ADDRESS,                  /* SOURCE BUFFER LENGTH */
            DBLEN ADDRESS,                  /* DEST BUFFER LENGTH */
            SBASE ADDRESS,                  /* SOURCE BUFFER BASE */
            /* THE VECTORS DBUFF AND SBUFF ARE DECLARED WITH DIMENSION
            1024, BUT ACTUALLY VARY WITH THE FREE MEMORY SIZE */
            DBUFF(1024) BYTE AT (.MEMORY), /* DESTINATION BUFFER */
            SBUFF BASED SBASE (1024) BYTE, /* SOURCE BUFFER */
            SDISK BYTE,                     /* SOURCE DISK */
            (SCOM, DHEX) BYTE,              /* SOURCE IS 'COM' FILE IF TRUE
          -   */
```

```
                                        /* DEST IS 'HEX' FILE IF TRUE
          -   */
              SOURCE (FSIZE) BYTE,      /* SOURCE FCB */
              SFUB BYTE AT(.SOURCE(13)),/* UNFILLED BYTES FIELD */
              DEST (FRSIZE) BYTE,       /* DESTINATION FCB */
              DESTR ADDRESS AT(.DEST(33)),/* RANDOM RECORD POSITION */
              DESTO BYTE    AT(.DEST(35)),/* RANDOM OVERFLOW BYTE */
              DFUB BYTE AT (.DEST(13)), /* UNFILLED BYTES FIELD */
              DDISK BYTE,               /* DESTINATION DISK */
              HBUFF(HBUFS) BYTE,        /* HEX FILE BUFFER */
              HSOURCE BYTE,             /* NEXT HEX SOURCE CHARACTER */

              NSOURCE ADDRESS,          /* NEXT SOURCE CHARACTER */
              HARDEOF ADDRESS,          /* SET TO NSOURCE ON REAL EOF */
              NDEST ADDRESS;            /* NEXT DESTINATION CHARACTER */

28  1       DECLARE
              /* SUBMIT FILE CONTROL BLOCK FOR ERROR DELETE */
              SUBFCB (*) BYTE DATA (0,'$$$     SUB',0,0,0);

29  1         DECLARE
              PDEST BYTE,               /* DESTINATION DEVICE */
              PSOURCE BYTE;             /* CURRENT SOURCE DEVICE */

30  1         DECLARE
              MULTCOM BYTE,             /* FALSE IF PROCESSING ONE LINE
          -   */
              PUTNUM BYTE,              /* SET WHEN READY FOR NEXT LINE
          -   NUM */
              CONCNT BYTE,               /* COUNTER FOR CONSOLE READY CH
          -   ECK */
              CHAR BYTE,                /* LAST CHARACTER SCANNED */
              TYPE BYTE,                /* TYPE OF CHARACTER SCANNED */
              FLEN BYTE;                /* FILE NAME LENGTH */

31  1       MON1: PROCEDURE(F,A) EXTERNAL;
32  2           DECLARE F BYTE,
              A ADDRESS;
33  2           END MON1;

34  1       MON2: PROCEDURE(F,A) BYTE EXTERNAL;
35  2           DECLARE F BYTE,
              A ADDRESS;
36  2           END MON2;

37  1       MON3: PROCEDURE(F,A) ADDRESS EXTERNAL;
38  2           DECLARE F BYTE,
              A ADDRESS;
39  2           END MON3;

40  1       BOOT: PROCEDURE EXTERNAL;
              /* SYSTEM REBOOT */
41  2           END BOOT;

42  1       READRDR: PROCEDURE BYTE;
              /* READ CURRENT READER DEVICE */
43  2           RETURN MON2(3,0);
```

```
44   2          END READRDR;

45   1      READCHAR: PROCEDURE BYTE;
               /* READ CONSOLE CHARACTER */
46   2          RETURN MON2(1,0);
47   2          END READCHAR;

48   1      DECLARE
               TRUE LITERALLY '1',
               FALSE LITERALLY '0',
               FOREVER LITERALLY 'WHILE TRUE',
               CR LITERALLY '13',
               LF LITERALLY '10',
               WHAT LITERALLY '63';

49   1      PRINTCHAR: PROCEDURE(CHAR);
50   2          DECLARE CHAR BYTE;
51   2          CALL MON1(2,CHAR AND 7FH);
52   2          END PRINTCHAR;

53   1      CRLF: PROCEDURE;
54   2          CALL PRINTCHAR(CR);
55   2          CALL PRINTCHAR(LF);
56   2          END CRLF;

57   1      PRINT: PROCEDURE(A);
58   2          DECLARE A ADDRESS;
               /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE
               NEXT DOLLAR SIGN IS ENCOUNTERED */
59   2          CALL CRLF;
60   2          CALL MON1(9,A);
61   2          END PRINT;

62   1      DECLARE DCNT BYTE;

63   1      VERSION: PROCEDURE ADDRESS;
64   2          RETURN MON3(12,0); /* VERSION NUMBER */
65   2          END VERSION;

66   1      INITIALIZE: PROCEDURE;
67   2          CALL MON1(13,0);
68   2          END INITIALIZE;

69   1      SELECT: PROCEDURE(D);
70   2          DECLARE D BYTE;
71   2          CALL MON1(14,D);
72   2          END SELECT;

73   1      OPEN: PROCEDURE(FCB);
74   2          DECLARE FCB ADDRESS;
75   2          DCNT = MON2(15,FCB);
76   2          END OPEN;

77   1      CLOSE: PROCEDURE(FCB);
78   2          DECLARE FCB ADDRESS;
79   2          DCNT = MON2(16,FCB);
80   2          END CLOSE;
```

```
 81   1      SEARCH: PROCEDURE(FCB);
 82   2          DECLARE FCB ADDRESS;
 83   2          DCNT = MON2(17,FCB);
 84   2          END SEARCH;

 85   1      SEARCHN: PROCEDURE;
 86   2          DCNT = MON2(18,0);
 87   2          END SEARCHN;

 88   1      DELETE: PROCEDURE(FCB);
 89   2          DECLARE FCB ADDRESS;
 90   2          CALL MON1(19,FCB);
 91   2          END DELETE;

 92   1      DISKREAD: PROCEDURE(FCB) BYTE;
 93   2          DECLARE FCB ADDRESS;
 94   2          RETURN MON2(20,FCB);
 95   2          END DISKREAD;

 96   1      DISKWRITE: PROCEDURE(FCB) BYTE;
 97   2          DECLARE FCB ADDRESS;
 98   2          RETURN MON2(21,FCB);
 99   2          END DISKWRITE;

100   1      MAKE: PROCEDURE(FCB);
101   2          DECLARE FCB ADDRESS;
102   2          DCNT = MON2(22,FCB);
103   2          END MAKE;

104   1      RENAME: PROCEDURE(FCB);
105   2          DECLARE FCB ADDRESS;
106   2          CALL MON1(23,FCB);
107   2          END RENAME;

108   1      DECLARE
              CUSER BYTE, /* CURRENT USER NUMBER */
              SUSER BYTE; /* SOURCE USER NUMBER ('G' PARAMETER) */

109   1      SETIND: PROCEDURE(FCB);
110   2          DECLARE FCB ADDRESS;
111   2          CALL MON1(30,FCB);
112   2          END SETIND;

113   1      GETUSER: PROCEDURE BYTE;
114   2          RETURN MON2(32,0FFH);
115   2          END GETUSER;

116   1      SETUSER: PROCEDURE(USER);
117   2          DECLARE USER BYTE;
118   2          CALL MON1(32,USER);
119   2          END SETUSER;

120   1      SETCUSER: PROCEDURE;
121   2          CALL SETUSER(CUSER);
122   2          END SETCUSER;
```

```
123   1       SETSUSER: PROCEDURE;
124   2          'CALL SETUSER(SUSER);
125   2           END SETSUSER;

126   1       READ$RANDOM: PROCEDURE(FCB) BYTE;
127   2           DECLARE FCB ADDRESS;
128   2           RETURN MON2(33,FCB);
129   2           END READ$RANDOM;

130   1       WRITE$RANDOM: PROCEDURE(FCB) BYTE;
131   2           DECLARE FCB ADDRESS;
132   2           RETURN MON2(34,FCB);
133   2           END WRITE$RANDOM;

134   1       SET$RANDOM: PROCEDURE(FCB);
135   2           DECLARE FCB ADDRESS;
                  /* SET RANDOM RECORD POSITION */
136   2           CALL MON1(36,FCB);
137   2           END SET$RANDOM;

138   1       DECLARE CBUFF(130) BYTE,   /* COMMAND BUFFER */
                    MAXLEN BYTE AT (.CBUFF(0)),  /* MAX BUFFER LENGTH */
                    COMLEN BYTE AT (.CBUFF(1)),  /* CURRENT LENGTH */
                    COMBUFF (128) BYTE AT (.CBUFF(2)); /* COMMAND BUFFER CONTENTS
            -  */
139   1       DECLARE (TCBP,CBP) BYTE;   /* TEMP CBP, COMMAND BUFFER POINTER */

140   1       READCOM: PROCEDURE;
                  /* READ INTO COMMAND BUFFER */
141   2           MAXLEN = 128;
142   2           CALL MON1(10,.MAXLEN);
143   2           END READCOM;

144   1       DECLARE MCBP BYTE;

145   1       CONBRK: PROCEDURE BYTE;
                  /* CHECK CONSOLE CHARACTER READY */
146   2           RETURN MON2(11,0);
147   2           END CONBRK;

148   1       DECLARE /* CONTROL TOGGLE VECTOR */
                    CONT(26) BYTE,   /* ONE FOR EACH ALPHABETIC */
                    /* 00 01 02 03 04 05 06 07 08 09 10 11 12 13
                        A  B  C  D  E  F  G  H  I  J  K  L  M  N
                       14 15 16 17 18 19 20 21 22 23 24 25
                        O  P  Q  R  S  T  U  V  W  X  Y  Z   */
                    BLOCK  BYTE  AT(.CONT(1)),     /* BLOCK MODE TRANSFER */
                    DELET  BYTE  AT(.CONT(3)),     /* DELETE CHARACTERS */
                    ECHO   BYTE  AT(.CONT(4)),     /* ECHO CONSOLE CHARACTERS */
                    FORMF  BYTE  AT(.CONT(5)),     /* FORM FILTER */
                    GETU   BYTE  AT(.CONT(6)),     /* GET FILE, USER # */
                    HEXT   BYTE  AT(.CONT(7)),     /* HEX FILE TRANSFER */
                    IGNOR  BYTE  AT(.CONT(8)),     /* IGNORE :00 RECORD ON FILE */
                    LOWER  BYTE  AT(.CONT(11)),    /* TRANSLATE TO LOWER CASE */
                    NUMB   BYTE  AT(.CONT(13)),    /* NUMBER OUTPUT LINES */
                    OBJ    BYTE  AT(.CONT(14)),    /* OBJECT FILE TRANSFER */
                    PAGCNT BYTE  AT(.CONT(15)),    /* PAGE LENGTH */
```

```
                QUITS  BYTE  AT(.CONT(16)),    /* QUIT COPY */
                RSYS   BYTE  AT(.CONT(17)),    /* READ SYSTEM FILES */
                STARTS BYTE  AT(.CONT(18)),    /* START COPY */
                TABS   BYTE  AT(.CONT(19)),    /* TAB SET */
                UPPER  BYTE  AT(.CONT(20)),    /* UPPER CASE TRANSLATE */
                VERIF  BYTE  AT(.CONT(21)),    /* VERIFY EQUAL FILES ONLY */
                WRROF  BYTE  AT(.CONT(22)),    /* WRITE TO R/O FILE */
                ZEROP  BYTE  AT(.CONT(25));    /* ZERO PARITY ON INPUT */

149   1         SETDMA: PROCEDURE(A);
150   2             DECLARE A ADDRESS;
151   2             CALL MON1(26,A);
152   2             END SETDMA;

              /* INTELLEC 8 INTEL/ICOM READER INPUT */

153   1       INTIN: PROCEDURE BYTE;
                  /* READ THE INTEL / ICOM READER */
154   2           DECLARE PTRI LITERALLY '3',  /* DATA */
                          PTRS LITERALLY '1',  /* STATUS */
                          PTRC LITERALLY '1',  /* COMMAND */
                          PTRG LITERALLY '0CH', /* GO */
                          PTRN LITERALLY '08H'; /* STOP */

                  /* STROBE THE READER */
155   2           OUTPUT(PTRC) = PTRG;
156   2           OUTPUT(PTRC) = PTRN;
157   2               DO WHILE NOT ROL(INPUT(PTRS),3); /* NOT READY */
158   3               END;
                  /* DATA READY */
159   2           RETURN INPUT(PTRI) AND 7FH;
160   2           END INTIN;


161   1       DECLARE ZEROSUP BYTE,  /* ZERO SUPPRESSION */
                  (C3,C2,C1) BYTE;     /* LINE COUNT ON PRINTER  */

162   1         ERROR: PROCEDURE(A);
163   2           DECLARE A ADDRESS, I BYTE;
164   2           CALL SETCUSER;
165   2           CALL PRINT(A); CALL PRINTCHAR(':'); CALL PRINTCHAR(' ');
168   2               DO I = TCBP TO CBP;
169   3               IF I < COMLEN THEN CALL PRINTCHAR(COMBUFF(I));
171   3               END;
                  /* ZERO THE COMLEN IN CASE THIS IS A SINGLE COMMAND */
172   2           COMLEN = 0;
                  /* DELETE ANY $$$.SUB FILES IN CASE BATCH PROCESSING */
                  /* DELETE SUB FILE ONLY IF PRESENT (MAY BE R/O DISK) */
173   2           CALL SEARCH(.SUBFCB);
174   2           IF DCNT <> 255 THEN CALL DELETE(.SUBFCB);
176   2           CALL CRLF;
177   2           GO TO RETRY;
178   2           END ERROR;

179   1         MOVE: PROCEDURE(S,D,N);
180   2           DECLARE (S,D) ADDRESS, N BYTE;
181   2           DECLARE A BASED S BYTE, B BASED D BYTE;
```

```
182   2             DO WHILE (N:=N-1) <> 255;
183   3             B = A; S = S+1; D = D+1;
186   3             END;
187   2           END MOVE;


188   1         FILLSOURCE: PROCEDURE;
                    /* FILL THE SOURCE BUFFERS */
189   2           DECLARE (I,J) BYTE;
190   2           NSOURCE = 0;
191   2           CALL SELECT(SDISK);
192   2           CALL SETSUSER; /* SOURCE USER NUMBER SET */
193   2               DO I = 0 TO NBUF;
                        /* SET DMA ADDRESS TO NEXT BUFFER POSIITION */
194   3               CALL SETDMA(.SBUFF(NSOURCE));
195   3               IF (J := DISKREAD(.SOURCE)) <> 0 THEN
196   3                   DO; IF J <> 1 THEN
198   4                       CALL ERROR(.('DISK READ ERROR$'));
                        /* END - OF - FILE */
199   4                   HARDEOF = NSOURCE; /* SET HARD END-OF-FILE */
200   4                   SBUFF(NSOURCE) = ENDFILE; I = NBUF;
202   4                   END; ELSE
203   3               NSOURCE = NSOURCE + 128;
204   3               END;
205   2           NSOURCE = 0;
206   2           CALL SETCUSER; /* BACK TO CURRENT USER NUMBER */
207   2           END FILLSOURCE;


208   1         WRITEDEST: PROCEDURE;
                    /* WRITE OUTPUT BUFFERS UP TO BUT NOT INCLUDING POSITION
                    NDEST - THE LOW ORDER 7 BITS OF NDEST ARE ZERO */
209   2           DECLARE (I, J, N) BYTE;
210   2           DECLARE DMA ADDRESS;
211   2           DECLARE DATAOK BYTE;
212   2           IF (N := LOW(SHR(NDEST,7)) - 1) = 255 THEN RETURN ;
214   2           NDEST = 0;
215   2           CALL SELECT(DDISK);
216   2           CALL SETRANDOM(.DEST); /* SET BASE RECORD FOR VERIFY */
217   2               DO I = 0 TO N;
                        /* SET DMA ADDRESS TO NEXT BUFFER */
218   3               DMA = .DBUFF(NDEST);
219   3               CALL SETDMA(DMA);
220   3               IF DISKWRITE(.DEST) <> 0 THEN
221   3                   CALL ERROR(.('DISK WRITE ERROR$'));
222   3               NDEST = NDEST + 128;
223   3               END;
224   2           IF VERIF THEN /* VERIFY DATA WRITTEN OK */
225   2               DO;
226   3               NDEST = 0;
227   3               CALL SETDMA(.BUFF); /* FOR COMPARE */
228   3                   DO I = 0 TO N;
229   4                   DATAOK = READRANDOM(.DEST) = 0;
230   4                   DESTR = DESTR + 1; /* NEXT RANDOM READ */
231   4                   J = 0;
                            /* PERFORM COMPARISON */
232   4                       DO WHILE DATAOK AND J < 80H;
```

```
233   5                     DATAOK = BUFFER(J) = DBUFF(NDEST+J);
234   5                     J = J + 1;
235   5                     END;
236   4                 NDEST = NDEST + 128;
237   4                 IF NOT DATAOK THEN
238   4                     CALL ERROR(.('VERIFY ERROR$'));
239   4                 END;
240   3             DATAOK = DISKWRITE(.DEST);
                    /* NOW READY TO CONTINUE THE WRITE OPE ATION */
241   3             END;
242   2           NDEST = 0;
243   2           END WRITEDEST;

244   1         PUTDCHAR: PROCEDURE(B);
245   2           DECLARE (B,IOB) BYTE;
                  /* WRITE BYTE B TO THE DESTINATION DEVICE GIVEN BY PDEST */
246   2           IF B >= ' ' THEN
247   2               DO; COLUMN = COLUMN + 1;
249   3               IF DELET > 0 THEN /* MAY BE PAST RIGHT SIDE */
250   3                   DO; IF COLUMN > DELET THEN RETURN;
253   4                   END;
254   3               END;
255   2           IOB = IOBYTE; /* IN CASE IT IS ALTERED */
256   2               DO CASE PDEST;
                        /* CASE 0 IS THE DESTINATION FILE */
257   3                   DO;
258   4                   IF NDEST >= DBLEN THEN CALL WRITEDEST;
260   4                   DBUFF(NDEST) = B;
261   4                   NDEST = NDEST+1;
262   4                   END;
                        /* CASE 1 IS ARD (ADDMASTER) */
263   3                   GO TO NOTDEST;
                       /* CASE 2 IS IRD (INTEL/ICOM) */
264   3                   GO TO NOTDEST;
                        /* CASE 3 IS PTR */
265   3                   GO TO NOTDEST;
                        /* CASE 4 IS UR1 */
266   3                   GO TO NOTDEST;
                        /* CASE 5 IS UR2 */
267   3                   GO TO NOTDEST;
                        /* CASE 6 IS RDR */
268   3                   NOTDEST:
                            CALL ERROR(.('NOT A CHARACTER SINK$'));
                        /* CASE 7 IS OUT */
269   3                   CALL OUT(B);
                        /* CASE 8 IS LPT */
270   3                       DO; IOBYTE = 1000$0000B; GO TO LSTL;
273   4                       END;
                        /* CASE 9 IS UL1 */
274   3                       DO; IOBYTE = 1100$0000B; GO TO LSTL;
277   4                       END;
                        /* CASE 10 IS PRN (TABS EXPANDED, LINES LISTED, CHANGED TO
      -     LST) */
278   3                       DO; IOBYTE = 1000$0000B; GO TO LSTL;
281   4                       END;
                        /* CASE 11 IS LST */
282   3                   LSTL:
```

```
                        CALL MON1(5,B);
                    /* CASE 12 IS PTP */
283   3                 DO; IOBYTE = 0001$0000B; GO TO PUNL;
286   4                 END;
                    /* CASE 13 IS UP1 */
287   3                 DO; IOBYTE = 0010$0000B; GO TO PUNL;
290   4                 END;
                    /* CASE 14 IS UP2 */
291   3                 DO; IOBYTE = 0011$0000B; GO TO PUNL;
294   4                 END;
                    /* CASE 15 IS PUN */
295   3                 PUNL:
                            CALL MON1(4,B);
                      /* CASE 16 IS TTY */
296   3                   DO; IOBYTE = 0; GO TO CONL;
299   4                   END;
                      /* CASE 17 IS CRT */
300   3                   DO; IOBYTE = 1; GO TO CONL;
303   4                   END;
                      /* CASE 18 IS UC1 */
304   3                   DO; IOBYTE = 11B; GO TO CONL;
307   4                   END;
                      /* CASE 19 IS CON */
308   3                   CONL:
                              CALL MON1(2,B);
309   3                 END;
310   2           IOBYTE = IOB;
311   2           END PUTDCHAR;

312   1       PUTDESTC: PROCEDURE(B);
313   2           DECLARE (B,I) BYTE;
                  /* WRITE DESTINATION CHARACTER, TAB EXPANSION */
314   2           IF B <> TAB THEN CALL PUTDCHAR(B); ELSE
316   2           IF TABS = 0 THEN CALL PUTDCHAR(B); ELSE
                      /* B IS TAB CHAR, TABS > 0 */
318   2               DO; I = COLUMN;
320   3                   DO WHILE I >= TABS;
321   4                   I = I - TABS;
322   4                   END;
323   3               I = TABS - I;
324   3                   DO WHILE I > 0;
325   4                   I = I - 1;
326   4                   CALL PUTDCHAR(' ');
327   4                   END;
328   3               END;
329   2           IF B = CR THEN COLUMN = 0;
331   2           END PUTDESTC;

332   1       PRINT1: PROCEDURE(B);
333   2           DECLARE B BYTE;
334   2           IF (ZEROSUP := ZEROSUP AND B = 0) THEN CALL PUTDESTC(' '); ELS
                E
336   2               CALL PUTDESTC('0'+B);
337   2           END PRINT1;

338   1       PRINTDIG: PROCEDURE(D);
339   2           DECLARE D BYTE;
```

```
340   2          CALL PRINT1(SHR(D,4)); CALL PRINT1(D AND 1111B);
342   2          END PRINTDIG;

343   1      NEWLINE: PROCEDURE;
344   2          DECLARE ONE BYTE;
345   2          ONE = 1;
346   2          ZEROSUP = NUMB = 1;
347   2          C1 = DEC(C1+ONE); C2 = DEC(C2 PLUS 0); C3 = DEC(C3 PLUS 0);
350   2          CALL PRINTDIG(C3); CALL PRINTDIG(C2); CALL PRINTDIG(C1);
353   2          IF NUMB = 1 THEN /* USUALLY PRINTER OUTPUT */
354   2              DO; CALL PUTDESTC(':'); CALL PUTDESTC(' ');
357   3              END; ELSE
358   2              CALL PUTDESTC(TAB);
359   2          END NEWLINE;

360   1      CLEARBUFF: PROCEDURE;
               /* CLEAR OUTPUT BUFFER IN BLOCK MODE TRANSMISION */
361   2          DECLARE NA ADDRESS;
362   2          DECLARE I BYTE;
363   2          I = LOW(NDEST) AND 7FH;  /* REMAINING PARTIAL BUFFER LENGTH */
364   2          NA = NDEST AND 0FF80H;   /* START OF SEGMENT NOT WRITTEN */
365   2          CALL WRITEDEST;          /* CLEARS BUFFERS */
366   2          CALL MOVE(.DBUFF(NA),.DBUFF,I);
               /* DATA MOVED TO BEGINNING OF BUFFER */
367   2          NDEST = I;
368   2          END CLEARBUFF;

369   1      PUTDEST: PROCEDURE(B);
370   2          DECLARE (I,B) BYTE;
               /* WRITE DESTINATION CHARACTER, CHECK TABS AND LINES */
371   2          IF FORMF THEN /* SKIP FORM FEEDS */
372   2              DO; IF B = FF THEN RETURN;
375   3              END;
376   2          IF PUTNUM THEN /* END OF LINE OR START OF FILE */
377   2              DO;
378   3              IF B <> FF THEN /* NOT FORM FEED */
379   3                  DO;
380   4                  IF (I:=PAGCNT) <> 0 THEN /* PAGE EJECT */
381   4                      DO; IF I=1 THEN I=LPP;
384   5                      IF (LINENO := LINENO + 1) >= I THEN
385   5                          DO; LINENO = 0; /* NEW PAGE */
387   6                          CALL PUTDESTC(FF);
388   6                          END;
389   5                      END;
390   4                  IF NUMB > 0 THEN
391   4                      CALL NEWLINE;
392   4                  PUTNUM = FALSE;
393   4                  END;
394   3              END;
395   2          IF BLOCK THEN /* BLOCK MODE TRANSFER */
396   2              DO;
397   3              IF B = XOFF AND PDEST = 0 THEN
398   3                  DO; CALL CLEARBUFF; /* BUFFERS WRITTEN */
400   4                  RETURN; /* DON'T PASS THE X-OFF */
401   4                  END;
402   3              END;
403   2          IF B = FF THEN LINENO = 0;
```

```
405   2          CALL PUTDESTC(B);
406   2          IF B = LF THEN PUTNUM = TRUE;
408   2          END PUTDEST;


409   1      UTRAN: PROCEDURE(B) BYTE;
410   2          DECLARE B BYTE;
                  /* TRANSLATE ALPHA TO UPPER CASE */
411   2          IF B >= 110$0001B AND B <= 111$1010B THEN /* LOWER CASE */
412   2              B = B AND 101$1111B; /* TO UPPER CASE */
413   2          RETURN B;
414   2          END UTRAN;

415   1      LTRAN: PROCEDURE(B) BYTE;
416   2          DECLARE B BYTE;
                  /* TRANSLATE TO LOWER CASE ALPHA */
417   2          IF B >= 'A' AND B <= 'Z' THEN B = B OR 10$0000B; /* TO LOWER *
          -   /
419   2          RETURN B;
420   2          END LTRAN;

421   1      GETSOURCEC: PROCEDURE BYTE;
                  /* READ NEXT SOURCE CHARACTER */
422   2          DECLARE (IOB,B,CONCHK) BYTE;

423   2          IF PSOURCE - 1 <= RDR THEN /* 1 ... RDR+1 */
424   2              DO; IF (BLOCK OR HEXT) AND CONBRK THEN
426   3                  DO;
427   4                  IF READCHAR = ENDFILE THEN RETURN ENDFILE;
429   4                  CALL PRINT(.('READER STOPPING',CR,LF,'$'));
430   4                  RETURN XOFF;
431   4                  END;
432   3              END;
433   2          CONCHK = TRUE; /* CONSOLE STATUS CHECK BELOW */
434   2          IOB = IOBYTE; /* SAVE IT IN CASE IT IS ALTERED */
435   2              DO CASE PSOURCE;
                      /* CASE 0 IS SOURCE FILE */
436   3                  DO; IF NSOURCE >= SBLEN THEN CALL FILLSOURCE;
439   4                  B = SBUFF(NSOURCE);
440   4                  NSOURCE = NSOURCE + 1;
441   4                  END;
                      /* CASE 1 IS INP */
442   3                  B = INP;
                      /* CASE 2 IS IRD (INTEL/ICOM) */
443   3                  B = INTIN;
                      /* CASE 3 IS PTR */
444   3                      DO; IOBYTE = 0000$0100B; GO TO RDRL;
447   4                      END;
                      /* CASE 4 IS UR1 */
448   3                      DO; IOBYTE = 0000$1000B; GO TO RDRL;
451   4                      END;
                      /* CASE 5 IS UR2 */
452   3                      DO; IOBYTE = 0000$1100B; GO TO RDRL;
455   4                      END;
                      /* CASE 6 IS RDR */
456   3                  RDRL:
                          B = MON2(3,0) AND 7FH;
```

```
                    /* CASE 7 IS OUT */
457   3                  GO TO NOTSOURCE;
                    /* CASE 8 IS LPT */
458   3                  GO TO NOTSOURCE;
                    /* CASE 9 IS UL1 */
459   3                  GO TO NOTSOURCE;
                    /* CASE 10 IS PRN */
460   3                  GO TO NOTSOURCE;
                    /* CASE 11 IS LST */
461   3                  GO TO NOTSOURCE;
                    /* CASE 12 IS PTP */
462   3                  GO TO NOTSOURCE;
                    /* CASE 13 IS UP1 */
463   3                  GO TO NOTSOURCE;
                    /* CASE 14 IS UP2 */
464   3                  GO TO NOTSOURCE;
                    /* CASE 15 IS PUN */
465   3                  NOTSOURCE:
                            DO; CALL ERROR(.('NOT A CHARACTER SOURCE$'));
467   4                         END;
                    /* CASE 16 IS TTY */
468   3                         DO; IOBYTE = 0; GO TO CONL;
471   4                         END;
                    /* CASE 17 IS CRT */
472   3                         DO; IOBYTE = 01B; GO TO CONL;
475   4                         END;
                    /* CASE 18 IS UC1 */
476   3                         DO; IOBYTE = 11B; GO TO CONL;
479   4                         END;
                    /* CASE 19 IS CON */
480   3                   CONL:
                            DO; CONCHK = FALSE; /* DON'T CHECK CONSOLE STATUS
          -   */
482   4                         B = MON2(1,0);
483   4                         END;
484   3                 END; /* OF CASES */
485   2               IOBYTE = IOB; /* RESTORE IOBYTE */
486   2               IF ECHO THEN /* COPY TO CONSOLE DEVICE */
487   2                   DO; IOB = PDEST; PDEST = CONP; CALL PUTDEST(B);
491   3                   PDEST = IOB;
492   3                   END;
493   2               IF CONCHK THEN /* TEST FOR CONSOLE CHAR READY */
494   2                   DO;
495   3                   IF SCOM THEN /* SOURCE IS A COM FILE */
496   3                       CONCHK = (CONCNT := CONCNT + 1) = 0; ELSE /* ASCII */
497   3                       CONCHK = B = LF;
498   3                   IF CONCHK THEN
499   3                       DO; IF CONBRK THEN
501   4                           DO;
502   5                           IF READCHAR = ENDFILE THEN RETURN ENDFILE;
504   5                           CALL ERROR(.('ABORTED$'));
505   5                           END;
506   4                       END;
507   3                   END;
508   2               IF ZEROP THEN B = B AND 7FH;
510   2               IF UPPER THEN RETURN UTRAN(B);
512   2               IF LOWER THEN RETURN LTRAN(B);
```

```
514   2          RETURN B;
515   2          END GETSOURCEC;

516   1      GETSOURCE: PROCEDURE BYTE;
                 /* GET NEXT SOURCE CHARACTER */
517   2          DECLARE CHAR BYTE;
518   2          MATCH: PROCEDURE(B) BYTE;
                     /* MATCH START AND QUIT STRINGS */
519   3              DECLARE (B,C) BYTE;
520   3              IF (C:=COMBUFF(B:=(B+MATCHLEN))) = ENDFILE THEN /* END MAT
          -    CH */
521   3                  DO; COMBUFF(B) = CHAR; /* SAVE CURRENT CHARACTER */
523   4                  RETURN TRUE;
524   4                  END;
525   3              IF C = CHAR THEN MATCHLEN = MATCHLEN + 1; ELS
527   3                  MATCH LEN = 0; /* NO MATCH */
528   3              RETURN FALSE;
529   3              END MATCH;
530   2          IF QUITLEN > 0 THEN
531   2              DO; IF (QUITLEN := QUITLEN - 1) = 1 THEN RETURN LF;
534   3              RETURN ENDFILE; /* TERMINATED WITH CR,LF,ENDFILE */
535   3              END;
536   2          DO FOREVER; /* LOOKING FOR START */
537   3          IF FEEDLEN > 0 THEN /* GET SEARCH CHARACTERS */
538   3              DO; FEEDLEN = FEEDLEN - 1;
540   4              CHAR = COMBUFF(FEEDBASE);
541   4              FEEDBASE = FEEDBASE + 1;
542   4              RETURN CHAR;
543   4              END;
544   3          IF (CHAR := GETSOURCEC) = ENDFILE THEN RETURN ENDFILE;
546   3          IF STARTS > 0 THEN /* LOOKING FOR START STRING */
547   3              DO; IF MATCH(STARTS) THEN
549   4                  DO; FEEDBASE = STARTS; STARTS = 0;
552   5                  FEEDLEN = MATCHLEN + 1;
553   5                  END; /* OTHERWISE NO MATCH, SKIP CHARACTER */
554   4              END; ELSE
555   3          IF QUITS > 0 THEN /* PASS CHARACTERS TIL MATCH */
556   3              DO; IF MATCH(QUITS) THEN
558   4                  DO; QUITS = 0; QUITLEN = 2;
                         /* SUBSEQUENTLY RETURN CR, LF, ENDFILE */
561   5                  RETURN CR;
562   5                  END;
563   4              RETURN CHAR;
564   4              END; ELSE
565   3          RETURN CHAR;
566   3          END; /* OF DO FOREVER */
567   2          END GETSOURCE;

568   1      DECLARE DISK BYTE;              /* SELECTED DISK */

569   1        GNC: PROCEDURE BYTE;
570   2          IF (CBP := CBP + 1) >= COMLEN THEN RETURN CR;
572   2          RETURN UTRAN(COMBUFF(CBP));
573   2          END GNC;

574   1        DEBLANK: PROCEDURE;
575   2            DO WHILE (CHAR := GNC) = ' ';
```

```
576   3             END;
577   2           END DEBLANK;

578   1         SCAN: PROCEDURE(FCBA);
579   2           DECLARE FCBA ADDRESS,          /* ADDRESS OF FCB TO FILL */
                  FCB BASED FCBA (FSIZE) BYTE;  /* FCB TEMPLATE */
580   2           DECLARE (I,J,K) BYTE; /* TEMP COUNTERS */

                /* SCAN LOOKS FOR THE NEXT DELIMITER, DEVICE NAME, OR FILE NAME.
                   THE VALUE OF CBP MUST BE 255 UPON ENTRY THE FIRST TIME */

581   2           DELIMITER: PROCEDURE(C) BYTE;
582   3               DECLARE (I,C) BYTE;
583   3               DECLARE DEL(*) BYTE DATA
                      (' =.:,<>',CR,LA,LB,RB);
584   3                   DO I = 0 TO LAST(DEL);
585   4                   IF C = DEL(I) THEN RETURN TRUE;
587   4                   END;
588   3               RETURN FALSE;
589   3               END DELIMITER;

590   2           PUTCHAR: PROCEDURE;
591   3               FCB(FLEN:=FLEN+1) = CHAR;
592   3               IF CHAR = WHAT THEN AMBIG = TRUE; /* CONTAINS AMBIGUOUS RE
          -   F */
594   3               END PUTCHAR;

595   2           FILLQ: PROCEDURE(LEN);
                      /* FILL CURRENT NAME OR TYPE WITH QUESTION MARKS */
596   3               DECLARE LEN BYTE;
597   3               CHAR = WHAT; /* QUESTION MARK */
598   3                   DO WHILE FLEN < LEN;
599   4                   CALL PUTCHAR;
600   4                   END;
601   3               END FILLQ;

602   2           GETFCB: PROCEDURE(I) BYTE;
603   3               DECLARE I BYTE;
604   3               RETURN FCB(I);
605   3               END GETFCB;

606   2           SCANPAR: PROCEDURE;
607   3               DECLARE (I,J) BYTE;
                      /* SCAN OPTIONAL PARAMETERS */
608   3               PARSET = TRUE;
609   3               SUSER = CUSER; /* SOURCE USER := CURRENT USER */
610   3               CHAR = GNC; /* SCAN PAST BRACKET */
611   3                   DO WHILE NOT(CHAR = CR OR CHAR = RB);
612   4                   IF (I := CHAR - 'A') > 25 THEN /* NOT ALPHA */
613   4                       DO; IF CHAR = ' ' THEN CHAR = GNC; ELSE
616   5                       CALL ERROR(.('BAD PARAMETER$'));
617   5                       END; ELSE
618   4                       DO; /* SCAN PARAMETER VALUE */
619   5                       IF CHAR = 'S' OR CHAR = 'Q' THEN
620   5                           DO; /* START OR QUIT COMMAND */
621   6                           J = CBP + 1; /* START OF STRING */
622   6                               DO WHILE NOT ((CHAR := GNC) = ENDFILE OR C
```

```
          -   HAR = CR);
 623   7                              END;
 624   6                           CHAR=GNC;
 625   6                          END; ELSE
 626   5                       IF (J := (CHAR := GNC) - '0') > 9 THEN J = 1;
                               ELSE
 628   5                           DO WHILE (K := (CHAR := GNC) - '0') <= 9;
 629   6                           J = J * 10 + K;
 630   6                           END;
 631   5                       CONT(I) = J;
 632   5                       IF I = 6 THEN /* SET SOURCE USER */
 633   5                           DO;
 634   6                           IF J > 31 THEN
 635   6                               CALL ERROR(.('INVALID USER NUMBER$'));
 636   6                           SUSER = J;
 637   6                           END;
 638   5                       END;
 639   4                   END;
 640   3               CHAR = GNC;
 641   3               END SCANPAR;

 642   2           CHKSET: PROCEDURE;
 643   3               IF CHAR = LA THEN CHAR = '=';
 645   3               END CHKSET;

                   /* INITIALIZE FILE CONTROL BLOCK TO EMPTY */
 646   2           AMBIG = FALSE; TYPE = ERR; CHAR = ' '; FLEN = 0;
 650   2               DO WHILE FLEN < FSIZE-1;
 651   3               IF FLEN = FNSIZE THEN CHAR = 0;
 653   3               CALL PUTCHAR;
 654   3               END;

                   /* DEBLANK COMMAND BUFFER */
 655   2               CALL DEBLANK;

                   /* SAVE STARTING POSITION OF SCAN FOR DIAGNOSTICS */
 656   2           TCBP = CBP;

                   /* MAY BE A SEPARATOR */
 657   2           IF DELIMITER(CHAR) THEN
 658   2               DO; CALL CHKSET;
 660   3               TYPE = SPECL; RETURN;
 662   3               END;

                   /* CHECK PERIPHERALS AND DISK FILES */
 663   2           DISK = 0;
                   /* CLEAR PARAMETERS */
 664   2               DO I = 0 TO 25; CONT(I) = 0;
 666   3               END;
 667   2           PARSET = FALSE;
 668   2           FEEDLEN,MATCHLEN,QUITLEN = 0;
                   /* SCAN NEXT NAME */
 669   2               DO FOREVER;
 670   3               FLEN = 0;
 671   3                   DO WHILE NOT DELIMITER(CHAR);
 672   4                   IF FLEN >= NSIZE THEN /* ERROR, FILE NAME TOO LONG */
 673   4                       RETURN;
```

```
674  4                  IF CHAR = '*' THEN CALL FILLQ(NSIZE); ELSE CALL PUTCHA
         -   R;
677  4                  CHAR = GNC;
678  4                  END;

                    /* CHECK FOR DISK NAME OR DEVICE NAME */
679  3                  IF CHAR = ':' THEN
680  3                      DO; IF DISK <> 0 THEN RETURN; /* ALREADY SET */
683  4                      IF FLEN = 1 THEN
                              /* MAY BE DISK NAME A ... Z */
684  4                          DO;
685  5                          IF (DISK := GETFCB(1) - 'A' + 1) > 26 THEN
686  5                          /* ERROR, INVALID DISK NAME */ RETURN;
687  5                          CALL DEBLANK; /* MAY BE DISK NAME ONLY */
688  5                          IF DELIMITER(CHAR) THEN
689  5                              DO; IF CHAR = LB THEN
691  6                                  CALL SCANPAR;
692  6                              CBP = CBP - 1;
693  6                              TYPE = DISKNAME;
694  6                              RETURN;
695  6                              END;
696  5                          END; ELSE

                          /* MAY BE A THREE CHARACTER DEVICE NAME */
697  4                      IF FLEN <> 3 THEN /* ERROR, CANNOT BE DEVICE NAME */
698  4                          RETURN; ELSE

                              /* LOOK FOR DEVICE NAME */
699  4                          DO; DECLARE (I,J,K) BYTE, M LITERALLY '20',
                              IO(*) BYTE DATA
                              ('INPIRDPTRUR1UR2RDROUTLPTUL1PRNLST',
                               'PTPUP1UP2PUNTTYCRTUC1CONNULEOF',0);
                              /* NOTE THAT ALL READER-LIKE DEVICES MUST BE
                              PLACED BEFORE 'RDR', AND ALL LISTING-LIKE DEVICES
                              MUST APPEAR BELOW LST, BUT ABOVE RDR.  THE LITERAL
                              DECLARATIONS FOR RDR, LST, AND PUNP MUST INDICATE
                              THE POSITIONS OF THESE DEVICES IN THE LIST */
701  5                          J = 255;
702  5                              DO K = 0 TO M;
703  6                              I = 0;
704  6                                  DO WHILE ((I:=I+1) <= 3) AND
                                        IO(J+I) = GETFCB(I);
705  7                                  END;
706  6                              IF I = 4 THEN /* COMPLETE MATCH */
707  6                                  DO; TYPE = PERIPH;
                                        /* SCAN PARAMETERS */
709  7                                  IF GNC = LB THEN CALL SCANPAR;
711  7                                  CBP = CBP - 1; CHAR = K;
713  7                                  RETURN;
714  7                                  END;
715  6                              /* OTHERWISE TRY NEXT DEVICE */ J = J + 3;
716  6                              END;

717  5                          /* ERROR, NO DEVICE NAME MATCH */ RETURN;
718  5                          END;
719  4                      IF CHAR = LB THEN /* PARAMETERS FOLLOW */
720  4                          CALL SCANPAR;
```

```
721   4                 END; ELSE

                      /* CHAR IS NOT ':', SO FILE NAME IS SET. SCAN REMAINDER */
722   3                 DO; IF FLEN = 0 THEN /* ERROR, NO PRIMARY NAME */
724   4                     RETURN;
725   4                 FLEN = FNAM;
726   4                 IF CHAR = '.' THEN /* SCAN FILE TYPE */
727   4                     DO WHILE NOT DELIMITER(CHAR := GNC);
728   5                     IF FLEN >= FNSIZE THEN
729   5                     /* ERROR, TYPE FIELD TOO LONG */ RETURN;
730   5                     IF CHAR = '*' THEN CALL FILLQ(FNSIZE);
732   5                         ELSE CALL PUTCHAR;
733   5                     END;

734   4                 IF CHAR = LB THEN
735   4                     CALL SCANPAR;
                        /* RESCAN DELIMITER NEXT TIME AROUND */
736   4                 CBP = CBP - 1;
737   4                 TYPE = FILE;
                        /* DISK IS THE SELECTED DISK (1 2 3 ... ) */
738   4                 IF DISK = 0 THEN DISK = CDISK + 1; /* DEFAULT */
740   4                 FCB(0),FCB(32) = 0;
741   4                 RETURN;
742   4                 END;
743   3             END;
744   2           END SCAN;

745   1         NULLS: PROCEDURE;
                    /* SEND 40 NULLS TO OUTPUT DEVICE */
746   2           DECLARE I BYTE;
747   2               DO I = 0 TO 39; CALL PUTDEST(0);
749   3               END;
750   2           END NULLS;


751   1         DECLARE FEXTH(FEXTL) BYTE,      /* HOLDS DESTINATION FILE TYPE *
          -     /
                    COPYING BYTE;               /* TRUE WHILE COPYING TO DEST FI
          -     LE */

752   1         MOVEXT: PROCEDURE(A);
753   2           DECLARE A ADDRESS;
                    /* MOVE THREE CHARACTER EXTENT INTO DEST FCB */
754   2           CALL MOVE(A,.DEST(FEXT),FEXTL);
755   2           END MOVEXT;

756   1       EQUAL: PROCEDURE(A,B) BYTE;
                    /* COMPARE THE STRINGS AT A AND B UNTIL EITHER A MISMATCH OR
                    A '$' IS ENCOUNTERED IN STRING B */
757   2           DECLARE (A,B) ADDRESS,
                    (SA BASED A, SB BASED B) BYTE;
758   2               DO WHILE SB <> '$';
759   3               IF (SB AND 7FH) <> (SA AND 7FH) THEN RETURN FALSE;
761   3               A = A + 1; B = B + 1;
763   3               END;
764   2           RETURN TRUE;
765   2           END EQUAL;
```

```
766   1       READ$EOF: PROCEDURE BYTE;
                  /* RETURN TRUE IF END OF FILE */
767   2           CHAR = GETSOURCE;
768   2          IF SCOM THEN RETURN HARDEOF < NSOURCE;
770   2           RETURN CHAR = ENDFILE;
771   2           END READ$EOF;


772   1       HEXRECORD: PROCEDURE BYTE;
                  /* READ ONE RECORD INTO SBUFF AND CHECK FOR PROPER FORM
                      RETURNS 0 IF RECORD OK
                      RETURNS 1 IF END OF TAPE (:00000)
                      RETURNS 2 IF ERROR IN RECORD       */


773   2           DECLARE XOFFSET BYTE; /* TRUE IF XOFF RECVD */
774   2           DECLARE NOERRS BYTE; /* TRUE IF NO ERRORS IN THIS RECORD */

775   2           PRINTERR: PROCEDURE(A);
                      /* PRINT ERROR MESSAGE IF NOERRS TRUE */
776   3               DECLARE A ADDRESS;
777   3               IF NOERRS THEN
778   3                   DO; NOERRS = FALSE;
780   4                   CALL PRINT(A);
781   4                   END;
782   3               END PRINTERR;

783   2           CHECKXOFF: PROCEDURE;
784   3               IF XOFFSET THEN
785   3                   DO; XOFFSET = FALSE;
787   4                   CALL CLEARBUFF;
788   4                   END;
789   3               END CHECKXOFF;

790   2           SAVECHAR: PROCEDURE BYTE;
                      /* READ CHARACTER AND SAVE IN BUFFER */
791   3               DECLARE I BYTE;
792   3               IF NOERRS THEN
793   3                   DO;
794   4                       DO WHILE (I := GETSOURCE) = XOFF; XOFFSET = TRUE;
796   5                       END;
797   4                   HBUFF(HSOURCE) = I;
798   4                   IF (HSOURCE := HSOURCE + 1) >= LAST(HBUFF) THEN
799   4                       CALL PRINTERR(.('RECORD TOO LONG$'));
800   4                   RETURN I;
801   4                   END;
802   3               RETURN ENDFILE; /* ON ERROR FLAG */
803   3               END SAVECHAR;

804   2           DECLARE (M, RL, CS, RT) BYTE,
                      LDA ADDRESS;  /* LOAD ADDRESS WHICH FOLLOWS : */

805   2          READHEX: PROCEDURE BYTE;
806   3               DECLARE H BYTE;
807   3               IF (H := SAVECHAR) - '0' <= 9 THEN RETURN H-'0';
809   3               IF H - 'A' > 5 THEN
```

```
 810   3                  CALL PRINTERR(.('INVALID DIGIT$'));
 811   3              RETURN H - 'A' + 10;
 812   3              END READHEX;

 813   2          READBYTE: PROCEDURE BYTE;
                        /* READ TWO HEX DIGITS */
 814   3              RETURN SHL(READHEX,4) OR READHEX;
 815   3              END READBYTE;

 816   2          READCS: PROCEDURE BYTE;
                        /* READ BYTE WITH CHECKSUM */
 817   3              RETURN CS := CS + READBYTE;
 818   3              END READCS;

 819   2          READADDR: PROCEDURE ADDRESS;
                        /* READ DOUBLE BYTE WITH CHECKSUM */
 820   3              RETURN SHL(DOUBLE(READCS),8) OR READCS;
 821   3              END READADDR;

 822   2          NOERRS = TRUE; /* NO ERRORS DETECTED IN THIS RECORD */

                    /* READ NEXT RECORD */
                        /* SCAN FOR THE ':' */
 823   2              HSOURCE = 0;
 824   2                  DO WHILE (CS := SAVECHAR) <> ':';
 825   3                  HSOURCE = 0;
 826   3                  IF CS = ENDFILE THEN
 827   3                      DO; CALL PRINT(.('END OF FILE, CTL-Z',WHAT,'$'));
 829   4                      IF READCHAR = ENDFILE THEN RETURN 1;
 831   4                          ELSE HSOURCE = 0;
 832   4                      END;
 833   3                  CALL CHECKXOFF;
 834   3                  END;

                        /* ':' FOUND */
 835   2                  CS = 0;
 836   2                  IF (RL := READCS) = 0 THEN /* END OF TAPE */
 837   2                      DO; DO WHILE (RL := SAVECHAR) <> ENDFILE;
 839   4                          CALL CHECKXOFF;
 840   4                          END;
 841   3                      IF NOERRS THEN RETURN 1;
 843   3                      RETURN 2;
 844   3                      END;

                    /* RECORD LENGTH IS NOT ZERO */
 845   2                  LDA = READADDR; /* LOAD ADDRESS */

                    /* READ WORDS UNTIL RECORD LENGTH EXHAUSTED */
 846   2                  RT = READCS; /* RECORD TYPE */
 847   2                     DO WHILE RL <> 0 AND NOERRS; RL = RL - 1;
 849   3                     M = READCS;
                           /* INCREMENT LA HERE FOR EXACT ADDRESS */
 850   3                     END;

                    /* CHECK SUM */
 851   2                  IF CS + READBYTE <> 0 THEN
 852   2                      CALL PRINTERR(.('CHECKSUM ERROR$'));
```

```
853   2          CALL CHECKXOFF;
854   2          IF NOERRS THEN RETURN 0;
856   2          RETURN 2;
857   2          END HEXRECORD;

858   1      READTAPE: PROCEDURE;
                 /* READ HEX FILE FROM HIGH SPEED READER TO 'HEX' FILE,
                 CHECK EACH RECORD FOR VALID DIGITS, AND PROPER CHECKSUM */
859   2          DECLARE (I,A) BYTE;
860   2              DO FOREVER;
861   3                  DO WHILE (I := HEXRECORD) <= 1;
862   4                  IF NOT (I = 1 AND IGNOR) THEN
863   4                      DO A = 1 TO HSOURCE;
864   5                      CALL PUTDEST(HBUFF(A-1));
865   5                      END;
866   4                  CALL PUTDEST(CR); CALL PUTDEST(LF);
868   4                  IF I = 1 THEN /* END OF TAPE ENCOUNTERED */
869   4                      RETURN;
870   4                  END;
871   3              CALL CRLF; HBUFF(HSOURCE) = '$';
873   3              CALL PRINT(.HBUFF);
874   3              CALL PRINT(.('CORRECT ERROR, TYPE RETURN OR CTL-Z$'));
875   3              CALL CRLF;
876   3              IF READCHAR = ENDFILE THEN RETURN;
878   3              END;
879   2          END READTAPE;

880   1      FORMERR: PROCEDURE;
881   2          CALL ERROR(.('INVALID FORMAT$'));
882   2          END FORMERR;

883   1      SETUPDEST: PROCEDURE;
884   2          CALL SELECT(DDISK);
885   2          DHEX = EQUAL(.DEST(FEXT),.('HEX$'));
886   2          CALL MOVE(.DEST(FEXT),.FEXTH,FEXTL); /* SAVE TYPE */
887   2          DEST(ROFILE) = DEST(ROFILE) AND 7FH;
888   2          DEST(SYSFILE)= DEST(SYSFILE)AND 7FH;
889   2          CALL MOVEXT(.('$$$'));
890   2          CALL DELETE(.DEST); /* REMOVE OLD $$$ FILE */
891   2          CALL MAKE(.DEST); /* CREATE A NEW ONE */
892   2          IF DCNT = 255 THEN CALL ERROR(.('NO DIRECTORY SPACE$'));
894   2          DEST(32),NDEST = 0;
895   2          END SETUPDEST;

896   1      SETUPSOURCE: PROCEDURE;
897   2          HARDEOF = 0FFFFH;
898   2          CALL SETSUSER; /* SOURCE USER */
899   2          CALL SELECT(SDISK);
900   2          CALL OPEN(.SOURCE);
901   2          CALL SETCUSER; /* BACK TO CURRENT USER */
902   2          IF (NOT RSYS) AND ROL(SOURCE(SYSFILE),1) THEN
903   2              DCNT = 255;
904   2          IF DCNT = 255 THEN CALL ERROR(.('NO FILE$'));
906   2          SOURCE(32) = 0;
                 /* CAUSE IMMEDIATE READ */
907   2          SCOM = EQUAL(.SOURCE(FEXT),.('COM$'));
```

```
 908   2          NSOURCE = SBLEN;
 909   2          END SETUPSOURCE;

 910   1      CHECK$STRINGS: PROCEDURE;
 911   2          IF STARTS > 0 THEN
 912   2              CALL ERROR(.('START NOT FOUND$'));
 913   2          IF QUITS  > 0 THEN
 914   2              CALL ERROR(.('QUIT NOT FOUND$'));
 915   2          END CHECK$STRINGS;

 916   1      CLOSEDEST: PROCEDURE(DIRECT);
 917   2          DECLARE DIRECT BYTE;
                   /* DIRECT IS TRUE IF SECTOR-BY-SECTOR COPY */
 918   2          IF DIRECT THEN
                       /* GET UNFILLED BYTES FROM SOURCE BUFFER */
 919   2              DFUB = SFUB; ELSE DFUB = 0;
 921   2              DO WHILE (LOW(NDEST) AND 7FH) <> 0;
 922   3              DFUB = DFUB + 1;
 923   3              CALL PUTDEST(ENDFILE);
 924   3              END;
 925   2          CALL CHECK$STRINGS;
 926   2          CALL WRITEDEST;
 927   2          CALL SELECT(DDISK);
 928   2          CALL CLOSE(.DEST);
 929   2          IF DCNT = 255 THEN
 930   2              CALL ERROR(.('CANNOT CLOSE DESTINATION FILE$'));
 931   2          CALL MOVEXT(.FEXTH); /* RECALL ORIGINAL TYPTE */
 932   2        DEST(12) = 0;
 933   2          CALL OPEN(.DEST);
 934   2          IF DCNT <> 255 THEN /* FILE EXISTS */
 935   2              DO;
 936   3              IF ROL(DEST(ROFILE),1) THEN /* READ ONLY */
 937   3                  DO;
 938   4                  IF NOT WRROF THEN
 939   4                      DO;
 940   5                      CALL PRINT (.('DESTINATION IS R/O, DELETE (Y/N)?$'
     -   ));
 941   5                      IF UTRAN(READCHAR) <> 'Y' THEN
 942   5                          DO; CALL PRINT(.('**NOT DELETED**$'));
 944   6                          CALL CRLF;
 945   6                          CALL MOVEXT(.('$$$'));
 946   6                          CALL DELETE(.DEST);
 947   6                          RETURN;
 948   6                          END;
 949   5                      CALL CRLF;
 950   5                      END;
 951   4                  DEST(ROFILE) = DEST(ROFILE) AND 7FH;
 952   4                  CALL SETIND(.DEST);
 953   4                  END;
 954   3              CALL DELETE(.DEST);
 955   3              END;
 956   2          CALL MOVE(.DEST,.DEST(16),16); /* READY FOR RENAME */
 957   2          CALL MOVEXT(.('$$$'));
 958   2          CALL RENAME(.DEST);
 959   2          END CLOSEDEST;

 960   1      SIZE$NBUF: PROCEDURE;
```

```
                    /* COMPUTE NUMBER OF BUFFERS - 1 FROM DBLEN */
 961    2           NBUF = (SHR(DBLEN,7) AND 0FFH) - 1;
                    /* COMPUTED AS DBLEN/128-1, WHERE DBLEN <= 32K (AND THUS
                    NBUF RESULTS IN A VALUE <= 2**15/2**7-1 = 2**8-1 = 255) */
 962    2           END SIZE$NBUF;

 963    1       SET$DBLEN: PROCEDURE;
                    /* ABSORB THE SOURCE BUFFER INTO THE DEST BUFFER */
 964    2           SBASE = .MEMORY;
 965    2           IF DBLEN >= 4000H THEN DBLEN = 7F80H; ELSE
 967    2              DBLEN = DBLEN + SBLEN;
 968    2           CALL SIZE$NBUF;
 969    2           END SET$DBLEN;

 970    1       SI E$MEMORY: PROCEDURE;
                    /* SET UP SOURCE AND DESTINATION BUFFERS */
 971    2           SBASE = .MEMORY + SHR(MEMSIZE - .MEMORY,1);
 972    2           SBLEN, DBLEN = SHR((MEMSIZE - .MEMORY) AND 0FF00H,1);
 973    2           CALL SIZE$NBUF;
 974    2           END SIZE$MEMORY;

 975    1       COPYCHAR: PROCEDURE;
                    /* PERFORM THE ACTUAL COPY FUNCTION */
 976    2           DECLARE RESIZED BYTE; /* TRUE IF SBUFF AND DBUFF COMBINED */
 977    2           IF (RESIZED := (BLOCK AND PSOURCE <> 0)) THEN /* BLOCK MODE */
 978    2               CALL SET$DBLEN; /* ABSORB SOURCE BUFFER */
 979    2           IF HEXT OR IGNOR THEN /* HEX FILE */
 980    2               CALL READTAPE; ELSE
 981    2               DO WHILE NOT READ$EOF;
 982    3               CALL PUTDEST(CHAR);
 983    3               END;
 984    2           IF RESIZED THEN
 985    2               DO; CALL CLEARBUFF;
 987    3               CALL SIZE$MEMORY;
 988    3               END;
 989    2           END COPYCHAR;

 990    1       SIMPLECOPY: PROCEDURE;
 991    2           DECLARE (FASTCOPY,I) BYTE;
 992    2           REAL$EOF: PROCEDURE BYTE;
 993    3               RETURN HARDEOF <> 0FFFFH;
 994    3               END REALEOF;
 995    2           CALL SIZE$MEMORY;
 996    2           TCBP = MCBP; /* FOR ERROR TRACING */
 997    2           CALL SETUPDEST;
 998    2           CALL SETUPSOURCE;
                    /* FILES READY FOR DIRECT COPY */
 999    2           FASTCOPY = TRUE;
                        /* LOOK FOR PARAMETERS */
1000    2           DO I = 0 TO 25;
1001    3           IF CONT(I) <> 0 THEN
1002    3               DO;
1003    4               IF NOT(I=6 OR I=14 OR I=17 OR I=21 OR I=22) THEN
                        /* NOT OBJ OR VERIFY */
1004    4               FASTCOPY = FALSE;
1005    4               END;
1006    3           END;
```

```
1007  2          IF FASTCOPY THEN /* COPY DIRECTLY TO DBUFF */
1008  2              DO; CALL SET$DBLEN; /* EXTEND DBUFF */
1010  3                  DO WHILE NOT REAL$EOF;
1011  4                  CALL FILLSOURCE;
1012  4                  IF REAL$EOF THEN
1013  4                      NDEST = HARDEOF; ELSE NDEST = DBLEN;
1015  4                  CALL WRITEDEST;
1016  4                  END;
1017  3              CALL SIZE$MEMORY; /* RESET TO TWO BUFFERS */
1018  3              END; ELSE
1019  2          CALL COPYCHAR;
1020  2          CALL CLOSEDEST(FASTCOPY);
1021  2          END SIMPLECOPY;

1022  1      MULTCOPY: PROCEDURE;
1023  2          DECLARE (NEXTDIR, NDCNT, NCOPIED) ADDRESS;
1024  2          PRNAME: PROCEDURE;
                    /* PRINT CURRENT FILE NAME */
1025  3              DECLARE (I,C) BYTE;
1026  3              CALL CRLF;
1027  3                  DO I = 1 TO FNSIZE;
1028  4                  IF (C := DEST(I)) <> ' ' THEN
1029  4                        DO; IF I = FEXT THEN CALL PRINTCHAR('.');
1032  5                        CALL PRINTCHAR(C);
1033  5                        END;
1034  4                  END;
1035  3              END PRNAME;

1036  2          NEXTDIR,NCOPIED = 0;
1037  2              DO FOREVER;
                    /* FIND A MATCHING ENTRY */
1038  3              CALL SETSUSER; /* SOURCE USER */
1039  3              CALL SELECT(SDISK);
1040  3              CALL SETDMA(.BUFFER);
1041  3              CALL SEARCH(.SEARFCB);
1042  3              NDCNT = 0;
1043  3                  DO WHILE (DCNT <> 255) AND NDCNT < NEXTDIR;
1044  4                  NDCNT = NDCNT + 1;
1045  4                  CALL SEARCHN;
1046  4                  END;
1047  3              CALL SETCUSER;
                    /* FILE CONTROL BLOCK IN BUFFER */
1048  3              IF DCNT = 255 THEN
1049  3                  DO; IF NCOPIED = 0 THEN
1051  4                  CALL ERROR(.('NOT FOUND$')); CALL CRLF;
1053  4                  RETURN;
1054  4                  END;
1055  3              NEXTDIR = NDCNT + 1;
                    /* GET THE FILE CONTROL BLOCK NAME TO DEST */
1056  3              CALL MOVE(.BUFFER+SHL(DCNT AND 11B,5),.DEST,16);
1057  3          DEST(0) = 0;
1058  3          DEST(12) = 0;
1059  3              CALL MOVE(.DEST,.SOURCE,16); /* FILL BOTH FCB'S */
1060  3              IF RSYS OR NOT ROL(DEST(SYSFILE),1) THEN /* OK TO READ */
1061  3                  DO;
1062  4                  IF (NCOPIED := NCOPIED + 1) = 1 THEN
1063  4                  CALL PRINT(.('COPYING -$'));
```

```
1064   4                    CALL PRNAME;
1065   4                    CALL SIMPLECOPY;
1066   4                    END;
1067   3                END;
1068   2            END MULTCOPY;

1069   1        SET$SDISK: PROCEDURE;
1070   2            IF DISK > 0 THEN SDISK = DISK - 1; ELSE SDISK = CDISK;
1073   2            END SET$SDISK;

1074   1        SET$DDISK: PROCEDURE;
1075   2            IF PARSET THEN /* PARAMETERS PRESENT */ CALL FORMERR;
1077   2            IF DISK > 0 THEN DDISK = DISK - 1; ELSE DDISK = CDISK;
1080   2            END SET$DDISK;

1081   1        CHECK$DISK: PROCEDURE;
1082   2            IF SUSER <> CUSER THEN /* DIFFERENT DISKS */
1083   2                RETURN;
1084   2            IF DDISK = SDISK THEN CALL FORMERR;
1086   2            END CHECK$DISK;

1087   1        CHECK$EOL: PROCEDURE;
1088   2            CALL DEBLANK;
1089   2            IF CHAR <> CR THEN CALL FORMERR;
1091   2            END CHECK$EOL;

1092   1        SCANDEST: PROCEDURE(COPYFCB);
1093   2            DECLARE COPYFCB ADDRESS;
1094   2            CALL SET$SDISK;
1095   2            CALL CHECK$EOL;
1096   2            CALL MOVE(.SOURCE,COPYFCB,33);
1097   2            CALL CHECK$DISK;
1098   2            END SCANDEST;

1099   1        SCANEQL: PROCEDURE;
1100   2            CALL SCAN(.SOURCE);
1101   2            IF NOT (TYPE = SPECL AND CHAR = '=') THEN CALL FORMERR;
1103   2            MCBP = CBP; /* FOR ERROR PRINTING */
1104   2            END SCANEQL;


1105   1        PIPENTRY:
                  /* BUFFER AT 80H CONTAINS REMAINDER OF LINE TYPED
                  FOLLOWING THE COMMAND 'PIP' - IF ZERO THEN PROMPT TIL CR */
                  CALL MOVE(.BUFF,.COMLEN,80H);
1106   1          MULTCOM = COMLEN = 0;

                  /* GET CURRENT CP/M VERSION */
1107   1          IF VERSION < CPMVERSION THEN
1108   1             DO;
1109   2             CALL PRINT(.('REQUIRES CP/M 2.0 OR NEWER FOR OPERATION.$'));
1110   2             CALL BOOT;
1111   2             END;
                  /* GET CURRENT USER */
1112   1          CUSER = GETUSER;
                  /* GET CURRENT DISK */
1113   1          CDISK = MON2(25,0);
```

```
1114   1        RETRY:
                /* ENTER HERE ON ERROR EXIT FROM THE PROCEDURE 'ERROR' */
                  CALL SIZE$MEMORY;
                  /* MAIN PROCESSING LOOP.  PROCESS UNTIL CR ONLY */
1115   1          DO FOREVER;
1116   2          SUSER = CUSER;
1117   2          C1, C2, C3 = 0; /* LINE COUNT = 000000 */
1118   2          PUTNUM = TRUE, /* ACTS LIKE LF OCCURRED ON ASCII FILE */
1119   2          CONCNT,COLUMN = 0; /* PRINTER TABS */
1120   2          LINENO = 254; /* INCREMENTED TO 255 > PAGCNT */
                  /* READ FROM CONSOLE IF NOT A ONELINER */
1121   2          IF MULTCOM THEN
1122   2              DO; CALL PRINTCHAR('*'); CALL READCOM;
1125   3              CALL CRLF;
1126   3              END;
1127   2          CBP = 255;
1128   2          IF COMLEN = 0 THEN /* SINGLE CARRIAGE RETURN */
1129   2              DO; CALL SELECT(CDISK);
1131   3              CALL BOOT;
1132   3              END;

                /* LOOK FOR SPECIAL CASES FIRST */
1133   2        DDISK,SDISK,PSOURCE,PDEST = 0;
1134   2          CALL SCAN(.DEST);
1135   2          IF TYPE = PERIPH THEN GO TO SIMPLECOM;
1137   2          IF TYPE = DISKNAME THEN
1138   2              DO; DDISK = DISK - 1;
1140   3              CALL SCANEQL;
1141   3              CALL SCAN(.SOURCE);
                      /* MAY BE MULTI COPY */
1142   3              IF TYPE <> FILE THEN CALL FORMERR;
1144   3              IF AMBIG THEN
1145   3                  DO; CALL SCANDEST(.SEARFCB);
1147   4                  CALL MULTCOPY;
1148   4                  END; ELSE
1149   3                  DO; CALL SCANDEST(.DEST);
                          /* FORM IS A:=B:UFN */
1151   4                  CALL SIMPLECOPY;
1152   4                  END;
1153   3              GO TO ENDCOM;
1154   3              END;

1155   2        IF TYPE <> FILE OR AMBIG THEN CALL FORMERR;
1157   2          CALL SET$DDISK;
1158   2          CALL SCANEQL;
1159   2          CALL SCAN(.SOURCE);
1160   2          IF TYPE = DISKNAME THEN
1161   2              DO;
1162   3              CALL SET$SDISK; CALL CHECK$DISK;
1164   3              CALL MOVE(.DEST,.SOURCE,33);
1165   3              CALL CHECK$EOL;
1166   3              CALL SIMPLECOPY;
1167   3              GO TO ENDCOM;
1168   3              END;
                  /* MAY BE POSSIBLE TO DO A FAST DISK COPY */
```

```
1169   2          IF TYPE = FILE THEN /* FILE TO FILE */
1170   2              DO; CALL DEBLANK; IF CHAR <> CR THEN GO TO SIMPLECOM;
                       /* FILE TO FILE */
1174   3              CALL SET$SDISK;
1175   3              CALL SIMPLECOPY;
1176   3              GO TO ENDCOM;
1177   3              END;

1178   2      SIMPLECOM:
                   CBP = 255; /* READY FOR RESCAN */

                   /* OTHERWISE PROCESS SIMPLE REQUEST */
1179   2          CALL SCAN(.DEST);
1180   2          IF (TYPE < FILE) OR AMBIG THEN /* DELIMITER OR ERROR */
1181   2              CALL ERROR(.('UNRECOGNIZED DESTINATION$'));

1182   2          DHEX = FALSE;
1183   2          IF TYPE = FILE THEN
1184   2              DO; /* DESTINATION IS A FILE, SAVE EXTENT NAME */
1185   3              CALL SET$DDISK;
1186   3              CALL SETUPDEST;
1187   3              CHAR = 255;
1188   3              END; ELSE
                   /* PERIPHERAL NAME */
1189   2          IF CHAR >= NULP OR CHAR <= RDR THEN CALL ERROR(.('CANNOT WRITE
           -   $'));

                   IF (PDEST := CHAR + 1) = PUNP THEN CALL NULLS;

                   /* NOW SCAN THE DELIMITER */
1193   2          CALL SCAN(.SOURCE);
1194   2          IF TYPE <> SPECL OR CHAR <> '=' THEN
1195   2              CALL ERROR(.('INVALID PIP FORMAT$'));

                   /* OTHERWISE SCAN AND COPY UNTIL CR */
1196   2          COPYING = TRUE;
1197   2              DO WHILE COPYING;
1198   3              SUSER = CUSER;
1199   3              CALL SCAN(.SOURCE);
                       /* SUSER MAY HAVE BEEN RESET */
1200   3              SCOM = FALSE;
1201   3              IF TYPE = FILE AND NOT AMBIG THEN /* A SOURCE FILE */
1202   3                  DO;
1203   4                  CALL SET$SDISK;
1204   4                  CALL SETUPSOURCE;
1205   4                  CHAR = 255;
1206   4                  END; ELSE

1207   3              IF TYPE <> PERIPH OR (CHAR <= LST AND CHAR > RDR) THEN
1208   3                  CALL ERROR(.('CANNOT READ$'));


                     SCOM = SCOM OR OBJ; /* MAY BE ABSOLUTE COPY */
1210   3             PSOURCE = CHAR + 1;
1211   3             IF CHAR = NULP THEN CALL NULLS; ELSE
1213   3             IF CHAR = EOFP THEN CALL PUTDEST(ENDFILE); ELSE
1215   3                 DO; /* DISK COPY */
```

```
1216   4                IF (CHAR < HSRDR AND DHEX) THEN HEXT = 1;
                        /* HEX FILE SET IF SOURCE IS RDR AND DEST IS HEX FILE */
1218   4                IF PDEST = PRNT THEN
1219   4                    DO; NUMB = 1;
1221   5                    IF TABS = 0 THEN TABS = 8;
1223   5                    IF PAGCNT = 0 THEN PAGCNT = 1;
1225   5                    END;
1226   4                CALL COPYCHAR;
1227   4                END;

1228   3              CALL CHECK$STRINGS;
                       /* READ ENDFILE, GO TO NEXT SOURCE */
1229   3              CALL SCAN(.SOURCE);
1230   3              IF TYPE <> SPECL OR (CHAR <> ',' AND CHAR <> CR) THEN
1231   3                  CALL ERROR(.('INVALID SEPARATOR$'));

1232   3              COPYING = CHAR <> CR;
1233   3              END;

                    /* IF NECESSARY, CLOSE FILE OR PUNCH TRAILER */
1234   2            IF PDEST = PUNP THEN
1235   2                DO; CALL PUTDEST(ENDFILE); CALL NULLS;
1238   3                END;
1239   2            IF PDEST = 0 THEN /* FILE HAS TO BE CLOSED AND RENAMED */
1240   2                CALL CLOSEDEST(FALSE);

                    /* COMLEN SET TO 0 IF NOT PROCESSING MULTIPLE COMMANDS */
1241   2         ENDCOM:
                      COMLEN = MULTCOM;

1242   2            END; /* DO FOREVER */
1243   1         END;
                  EOF
```



MODULE INFORMATION:

```
     CODE AREA SIZE     = 1C34H   7220D
     VARIABLE AREA SIZE = 01D8H    472D
     MAXIMUM STACK SIZE = 0034H     52D
     1594 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
0000 PIP#
0000 PIPMOD#
07E6 14         07EA 16         07F2 17         07F3 18         07F3 19
07FB 20         07FF 21         07FF 22         07FF 23         0804 24
0809 25         080A 42         080A 43         0813 44         0813 45
0813 46         081C 47         081C 49         0820 51         082D 52
082E 53         082E 54         0833 55         0838 56         0839 57
083F 59         0842 60         084B 61         084C 63         084C 64
0855 65         0855 66         0855 67         085D 68         085E  9
0862 71         086D 72         086E 73         0874 75         0880 76
0881 77         0887 79         0893 80         0894 81         089A 83
08A6 84         08A7 85         08A7 86         08B2 87         08B3 88
08B9 90         08C2 91         08C3 92         08C9 94         08D3 95
08D3 96         08D9 98         08E3 99         08E3 100        08E9 102
08F5 103        08F6 104        08FC 106        0905 107        0906  09
090C 111        0915 112        0916 113        0916 114        091F 115
091F 116        0923 118        092E 119        092F 120        092F 121
0936 122        0937 123        0937 124        093E 125        093F 126
0945 128        094F 129        094F 130        0955 132        095F 133
095F 134        0965 136        096E 137        096F 140        096F 141
0974 142        097C 143        097D 145        097D 146        0986 147
0986 149        098C 151        0995 152        0996 153        0996 155
099A 156        099E 157        09A7 158        09AA 159        09AF 160
09AF 162        09B5 164        09B8 165        09C0 166        09C5 167
09CA 168        09DA 169        09E4 170        09F1 171        09F8 172
09FD 173        0A03 174        0A0B 175        0A11 176        0A14 177
0A17 178        0A18 179        0A27 182        0A33 183        0A3D 184
0A44 185        0A4B 186        0A4E 187        0A4F 188        0A4F 190
0A55 191        0A5C 192        0A5F 193        0A6E 194        0A7B 195
0A89 197        0A91 198        0A97 199        0A9D 200        0AA4 201
0AAA 202        0AAD 203        0AB7 204        0ABE 205        0AC4 206
0AC7 207        0AC8 208        0AC8 212        0ADA 213        0ADB 214
0AE1 215        0AE8 216        0AEE 217        0AFD 218        0B07 219
0B0F 220        0B1A 221        0B20 222        0B2A 223        0B31 224
0B38 226        0B3E 227        0B44 228        0B53 229        0B61 230
0B68 231        0B6D 232        0B7B 233        0B9B 234        0B9F 235
0BA2 236        0BAC 237        0BB3 238        0BB9 239        0BC0 240
0BC9 241        0BC9 242        0BCF 243        0BD0 244        0BD4 246
0BDC 248        0BE0 249        0BE9 251        0BF3 252        0BF4 253
0BF4 254        0BF4 255        0BFA 256        0C0A 257        0C0A 258
0C16 259        0C19 260        0C24 261        0C2B 262        0C2E 263
0C31 264        0C34 265        0C37 266        0C3A 267        0C3D 268
0C46 269        0C50 270        0C50 271        0C55 272        0C58 273
0C5B 274        0C5B 275        0C60 276        0C63 277        0C66 278
0C66 279        0C6B 280        0C6E 281        0C71 282        0C7F 283
0C7F 284        0C84 285        0C87 286        0C8A 287        0C8A 288
0C8F 289        0C92 290        0C95 291        0C95 292        0C9A 293
0C9D 294        0CA0 295        0CAE 296        0CAE 297        0CB3 298
0CB6 299        0CB9 300        0CB9 301        0CBE 302        0CC1 303
0CC4 304        0CC4 305        0CC9 306        0CCC 307        0CCF 308
0CDD 309        0D05 310        0D0B 311        0D0C 312        0D10 314
0D18 315        0D22 316        0D2A 317        0D34 319        0D3A 320
0D44 321        0D4E 322        0D51 323        0D59 324        0D62 325
0D66 326        0D6B 327        0D6E 328        0D6E 329        0D76 330
0D7B 331        0D7C 332        0D80 334        0D91 335        0D99 336
```

0DA2 337
0DA3 338
0DA7 340
0DB4 341
0DBD 342
0DBE 343
0DBE 345
0DC3 346
0DCE 347
0DD6 348
0DDF 349
0DE8 350
0DEF 351
0DF6 352
0DFD 353
0E05 355
0E0A 356
0E0F 357
0E12 358
0E17 359
0E18 360
0E18 363
0E21 364
0E2A 365
0E2D 366
0E3C 367
0E44 368
0E45 369
0E49 371
0E50 373
0E58 374
0E59 375
0E59 376
0E60 378
0E68 380
0E73 382
0E7B 383
0E80 384
0E8E 386
0E93 387
0E98 388
0E98 389
0E98 390
0EA1 391
0EA4 392
0EA9 393
0EA9 394
0EA9 395
0EB0 397
0EC8 399
0ECB 400
0ECC 401
0ECC 402
0ECC 403
0ED4 404
0ED9 405
0EE0 406
0EE8 407
0EED 408
0EEE 409
0EF2 411
0F09 412
0F11 413
0F15 414
0F15 415
0F19 417
0F30 418
0F38 419
0F3C 420
0F3C 421
0F3C 423
0F47 425
0F59 427
0F61 428
0F64 429
0F6A 430
0F6D 431
0F6D 432
0F6D 433
0F72 434
0F78 435
0F88 436
0F88 437
0F94 438
0F97 439
0FA3 440
0FAA 441
0FAD 442
0FB6 443
0FBF 444
0FBF 445
0FC4 446
0FC7 447
0FCA 448
0FCA 449
0FCF 450
0FD2 451
0FD5 452
0FD5 453
0FDA 454
0FDD 455
0FE0 456
0FF0 457
0FF3 458
0FF6 459
0FF9 460
0FFC 461
0FFF 462
1002 463
1005 464
1008 465
1008 466
100E 467
1011 468
1011 469
1016 470
1019 471
101C 472
101C 473
1021 474
1024 475
1027 476
1027 477
102C 478
102F 479
1032 480
1032 481
1037 482
1042 483
1045 484
106D 485
1073 486
107A 488
1080 489
1085 490
108C 491
1092 492
1092 493
1099 495
10A0 496
10B2 497
10BD 498
10C4 500
10CB 502
10D3 503
10D6 504
10DC 505
10DC 506
10DC 507
10DC 508
10E3 509
10EB 510
10F2 511
10FA 512
1101 513
1109 514
110D 515
110D 516
11AD 518
11B1 520
11C9 522
11D6 523
11D9 524
11D9 525
11E3 526
11EA 527
11EF 528
11F2 529
110D 530
1116 532
1122 533
1125 534
1128 535
1128 536
1128 537
1131 539
1135 540
1141 541
1145 542
1146 543
1146 544
1151 545
1154 546
115D 548
1168 550
116E 551
1173 552
117A 553
117A 554
117D 555
1186 557
1191 559
1196 560
119B 561
119E 562
119E 563
11A2 564
11A5 565
11A9 566
11AC 567
11F2 569
11F2 570
1200 571
1203 572
1211 573
1211 574
1211 575
121C 576
121F 577
1220 578
1438 581
143C 584
144A 585
145A 586
145D 587
1464 588
1467 589
1467 590
1467 591
1479 592
1481 593
1486 594
1487 595
14?B 597
1490 598
149A 599
149D 600
14A0 601
14A1 602
14A5 604
14B1 605
14B1 606
14B1 608
14B6 609
14BC 610
14C2 611
14DA 612
14E9 614
14F1 615
14FA 616
1500 617
1503 619
151B 621
1522 622
153D 623
1540 624
1546 625
1549 626
155B 627
1563 628
1575 629
158A 630
158D 631
159A 632
15A2 634
15AB 635
15B1 636
15B7 637
15B7 638
15B7 639
15BA 640
15C0 641
15C1 642
15C1 643
15C9 644
15CE 645
1226 646
122B 647
1230 648
1233 649
1238 650
1240 651
1248 652
124D 653
1250 654

1253 655
1256 656
125C 657
1267 659
126A 660
126F 661
1270 662
1270 663
1275 664
1283 665
128E 666
1295 667
129A 668
12A5 669
12A5 670
12AA 671
12B5 672
12BD 673
12BE 674
12C6 675
12CE 676
12D1 677
12D7 678
12DA 679
12E2 681
12EA 682
12EB 683
12F3 685
1305 686
1306 687
1309 688
1314 690
131C 691
131F 692
1323 693
1328 694
1329 695
1329 696
132C 697
1334 698
1335 701
133A 702
1346 703
134B 704
137B 705
137E 706
1386 708
138B 709
1393 710
1396 711
139A 712
13A0 713
13A1 714
13A1 715
13A9 716
13B0 717
13B1 718
13B1 719
13B9 720
13BC 721
13BF 723
13C7 724
13C8 725
13CD 726
13D5 727
13E3 728
13EB 729
13EC 730
13F4 731
13FC 732
13FF 733
1402 734
140A 735
140D 736
1411 737
1416 738
141E 739
1425 740
1433 741
1434 742
1434 743
1437 744
15CF 745
15CF 747
15DD 748
15E2 749
15E9 750
15EA 752
15F0 754
15FC 755
15FD 756
1607 758
1610 759
1623 760
1626 761
162D 762
1634 763
1637 764
163A 765
163A 766
163A 767
1640 768
1647 769
1652 770
165B 771
165B 772
1712 775
1718 777
171F 779
1724 780
172C 781
172C 782
172D 783
172D 784
1734 786
1739 787
173C 788
173C 789
173D 790
173D 792
1744 794
174F 795
1754 796
1757 797
1764 798
1770 799
1776 800
177A 801
177A 802
177D 803
177D 805
177D 807
178C 808
1792 809
179E 810
17A4 811
17AC 812
17AC 813
17AC 814
17BB 815
17BB 816
17BB 817
17C4 818
17C4 819
17C4 820
17DA 821
165B 822
1660 823
1665 824
1670 825
1675 826
167D 828
1683 829
168B 830
168E 831
1693 832
1693 833
1696 834
1699 835
169E 836
16A9 838
16B4 839
16B7 840
16BA 841
16C1 842
16C4 843
16C7 844
16C7 845
16CD 846
16D3 847
16E3 848
16E7 849
16ED 850
16F0 851
16FC 852
1702 853
1705 854
170C 855
170F 856
1712 857
17DA 858
17DA 860
17DA 861
17E7 862
17F7 863
1806 864
1815 865
181C 866
1821 867
1826 868
182E 869
182F 870
1832 871
1835 872
1840 873
1846 874
184C 875
184F 876
1857 877
1858 878
185B 879
185C 880
185C 881
1862 882
1863 883
1863 884
186A 885
1876 886
1882 887
188A 888
1892 889
1898 890
189E 891
18A4 892
18AC 893
18B2 894
18BD 895
18BE 896
18BE 897
18C4 898
18C7 899
18CE 900
18D4 901
18D7 902
18E7 903
18EC 904
18F4 905
18FA 906
18FF 907
190B 908
1911 909
1912 910
1912 911
191B 912
1921 913
192A 914
1930 915
1931 916
1935 918
193C 919
1945 920
194A 921
1955 922
1959 923
195E 924
1961 925
1964 926
1967 927
196E 928
1974 929
197C 930
1982 931
1988 932
198D 933
1993 934
199B 936
19A3 938
19AA 940
19B0 941
19BC 943
19C2 944
19C5 945
19CB 946
19D1 947
19D2 948
19D2 949
19D5 950
19D5 951
19DD 952
19E3 953
19E3 954
19E9 955
19E9 956
19F5 957
19FB 958

```
1A01 959        1A02 960        1A02 961        1A15 962        1A16 963
1A16 964        1A1C 965        1A28 966        1A31 967        1A3C 968
1A3F 969        1A40 970        1A40 971        1A56 972        1A68 973
1A6B 974        1A6C 975        1A6C 977        1A7F 978        1A82 979
1A8D 980        1A93 981        1A9A 982        1AA1 983        1AA4 984
1AAB 986        1AAE 987        1AB1 988        1AB1 989        1AB2 990
1B6A 992        1B6A 993        1B78 994        1AB2 995        1AB5 996
1ABB 997        1ABE 998        1AC1 999        1AC6 1000       1AD2 1001
1AE1 1003       1B D 1004       1B22 1005       1B22 1006       1B29 1007
1B30 1009       1B33 1010       1B3A 1011       1B3D 1012       1B44 1013
1B4D 1014       1B53 1015       1B56 1016       1B59 1017       1B5C 1018
1B5F 1019       1B62 1020       1B69 1021       1B78 1022       1C49 1024
1C49 1026       1C4C 1027       1C5A 1028       1C6C 1030       1C74 1031
1C79 1032       1C80 1033       1C80 1034       1C87 1035       1B78 1036
1B81 037        1B 1 1038       1B84 1039       1B8B 1040       1B91 1041
1B97 1042       1B9D 1043       1BB7 1044       1BBE 1045       1BC1 1046
1BC4 1047       1BC7 1048       1BCF 1050       1BDB 1051       1BE1 1052
1BE4 1053       1BE5 1054       1BE5 1055       1BEC 1056       1C06 1057
1C0B 1058       1C10 1059       1C1C 1060       1C29 1062       1C39 1063
1C3F 1064       1C42 1065       1C45 1066       1C45 1067       1C48 1068
1C88 1069       1C88 1070       1C91 1071       1C9B 1072       1CA1 1073
1CA2 1074       1CA2 1075       1CA9 1076       1CAC 1077       1CB5 1078
1CBF 1079       1CC5 1080       1CC6 1081       1CC6 1082       1CD0 1083
1CD1 1084       1CDB 1085       1CDE 1086       1CDF 1087       1CDF 1088
1CE2 1089       1CEA 1090       1CED 1091       1CEE 1092       1CF4 1094
1CF7 1095       1CFA 1096       1D08 1097       1D0B 1098       1D0C 1099
1D0C 1100       1D12 1101       1D2A 1102       1D2D 1103       1D33 1104
04CE 1105       04DD 1106       04E8 1107       04F4 1109       04FA 1110
04FD 1111       04FD 1112       0503 1113       050E 1114       0514 1115
0514 1116       051A 1117       0525 1118       052A 1119       0532 1120
0535 1121       053C 1123       0541 1124       0544 1125       0547 1126
0547 1127       054C 1128       0554 1130       055B 1131       055E 1132
055E 1133       0570 1134       0576 1135       057E 1136       0581 1137
0589 1139       0590 1140       0593 1141       0599 1142       05A1 1143
05A4 1144       05AB 1146       05B1 1147       05B4 1148       05B7 1150
05BD 1151       05C0 1152       05C0 1153       05C3 1154       05C3 1155
05D3 1156       05D6 1157       05D9 1158       05DC 1159       05E2 1160
05EA 1162       05ED 1163       05F0 1164       05FC 1165       05FF 1166
0602 1167       0605 1168       0605 1169       060D 1171       0610 1172
0618 1173       061B 1174       061E 1175       0621 1176       0624 1177
0624 1178       0629 1179       062F 1180       063D 1181       0643 1182
0648 1183       0650 1185       0653 1186       0656 1187       065B 1188
065E 1189       0675 1190       067B 1191       0687 1192       068A 1193
0690 1194       06A8 1195       06AE 1196       06B3 1197       06BA 1198
06C0 1199       06C6 1200       06CB 1201       06DF 1203       06E2 1204
06E5 1205       06EA 1206       06ED 1207       070D 1208       0713 1209
071B 1210       0722 1211       072A 1212       0730 1213       0738 1214
0740 1216       074E 1217       0753 1218       075B 1220       0760 1221
0768 1222       076D 1223       0775 1224       077A 1225       077A 1226
077D 1227       077D 1228       0780 1229       0786 1230       07AA 1231
07B0 1232       07BB 1233       07BE 1234       07C6 1236       07CB 1237
07CE 1238       07CE 1239       07D6 1240       07DB 1241       07E1 1242
07E4 1243
0000 MODULE#
```