DIGITAL RESEARCH

Post Office Box 579, Pacific Grove, California 93950, (408) 373-3403

PERIPHERAL INTERCHANGE PROGRAM (PIP)

CP/M VERSION _____

SER. # _____

```
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

                                COPYRIGHT (C) DIGITAL RESEARCH
                                      NOVEMBER, 1976
                                                                    */
2   1       DECLARE COPYRIGHT(*) BYTE DATA (
            '   COPYRIGHT (C) 1976, DIGITAL RESEARCH.  PIP VERS 1.5');

3   1       DECLARE INPLOC ADDRESS DATA (103H);  /* ADDRESS OF INP. DEVICE */
4   1       DECLARE OUTLOC ADDRESS DATA (106H);  /* ADDRESS OF OUT. DEVICE */

5   1   OUT: PROCEDURE(B);
6   2       DECLARE B BYTE;
            /* SEND B TO OUT. DEVICE */
7   2       CALL OUTLOC;
8   2       END OUT;

9   1   INP: PROCEDURE BYTE;
10  2       CALL INPLOC;
            /* PATCHED TO RETURN REG-A */
11  2       RETURN 0;
12  2       END INP;


13  1   TIMEOUT: PROCEDURE;
            /* WAIT FOR 50 MSEC */
14  2       CALL TIME(250); CALL TIME(250);
16  2       END TIMEOUT;

          /* LITERAL DECLARATIONS */
17  1       DECLARE
            LIT LITERALLY 'LITERALLY',
            ENDFILE LIT '1AH',
            TAB LIT '9H',
            LA LIT '5FH',
            LB LIT '5BH',   /* LEFT BRACKET */
            RB LIT '5DH',   /* 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',
            NSIZE LIT '8',
            FNSIZE LIT '11',
            NDISK LIT '1',
            FNAM LIT '8',
            FEXT LIT '9',
            FEXTL LIT '3',
            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';

10  1       DECLARE
              EXIT ADDRESS INITIAL(0H),     /* REBOOT ADDRESS UPON COMPLETION */
              COLUMN BYTE,      /* COLUMN COUNT FOR PRINTER TABS */
              AMBIG BYTE,                   /* SET FOR AMBIGUOUS FILE REFS */
              PARSET BYTE,                  /* TRUE IF PARAMETERS PRESENT */
              FEEDBASE BYTE,                /* USED TO FEED SEARCH CHARACTERS */
              FEEDLEN BYTE,                 /* LENGTH OF FEED STRING */
              MATCHLEN BYTE,                /* USED IN MATCHING STRINGS */
              QUITLEN BYTE,                 /* USED TO TERMINATE QUIT COMMAND */
              NBUF BYTE,                    /* NUM BUFFERS-1 IN SBUFF AND DBUFF */
              CDISK BYTE,                   /* CURRENT DISK */
              BUFFER (128) BYTE AT (80H),   /* DEFAULT BUFFER */
              SEARFCB (FSIZE) BYTE AT (5CH), /* SEARCH FCB IN MULTI COPY */
              MEMSIZE ADDRESS AT (6H),      /* 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 */
              DEST (FSIZE) BYTE,            /* DESTINATION FCB */
              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 */

19  1       DECLARE
              PDEST BYTE,                   /* DESTINATION DEVICE */
              PSOURCE BYTE;                 /* CURRENT SOURCE DEVICE */

20  1       DECLARE
              MULTCOM BYTE,                 /* FALSE IF PROCESSING ONE LINE */
              PUTNUM BYTE,                  /* SET WHEN READY FOR NEXT LINE NUM */
              CONCNT BYTE,                   /* COUNTER FOR CONSOLE READY CHECK */
              CHAR BYTE,                    /* LAST CHARACTER SCANNED */
              TYPE BYTE,                    /* TYPE OF CHARACTER SCANNED */
              FLEN BYTE;                    /* FILE NAME LENGTH */

          $INCLUDE(:F1:CPIO.PLB)
21  1   =   MON1: PROCEDURE(F,A);
22  2   =       DECLARE F BYTE,
            =       A ADDRESS;
23  2   =       L1: GO TO L1; /* PATCHED WITH JMP 0005 */
```

```
24   2  =      END MON1;
        =
25   1  =  MON2: PROCEDURE(F,A) BYTE;
26   2  =      DECLARE F BYTE,
        =          A ADDRESS;
27   2  =      L2: GO TO L2; /* PATCHED WITH JMP 0005 */
28   2  =      RETURN 0;
29   2  =      END MON2;
        =
30   1  =  READRDR: PROCEDURE BYTE;
        =      /* READ CURRENT READER DEVICE */
31   2  =      RETURN MON2(3,0);
32   2  =      END READRDR;
        =
33   1  =  READCHAR: PROCEDURE BYTE;
        =      /* READ CONSOLE CHARACTER */
34   2  =      RETURN MON2(1,0);
35   2  =      END READCHAR;
        =
36   1  =  DECLARE
        =          TRUE LITERALLY '1',
        =          FALSE LITERALLY '0',
        =          FOREVER LITERALLY 'WHILE TRUE',
        =          CR LITERALLY '13',
        =          LF LITERALLY '10',
        =          WHAT LITERALLY '63';
        =
37   1  =  PRINTCHAR: PROCEDURE(CHAR);
38   2  =      DECLARE CHAR BYTE;
39   2  =      CALL MON1(2,CHAR);
40   2  =      END PRINTCHAR;
        =
41   1  =  CRLF: PROCEDURE;
42   2  =      CALL PRINTCHAR(CR);
43   2  =      CALL PRINTCHAR(LF);
44   2  =      END CRLF;
        =
45   1  =  PRINT: PROCEDURE(A);
46   2  =      DECLARE A ADDRESS;
        =      /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE
        =      NEXT DOLLAR SIGN IS ENCOUNTERED */
47   2  =      CALL CRLF;
48   2  =      CALL MON1(9,A);
49   2  =      END PRINT;
        =
50   1  =  DECLARE DCNT BYTE;
        =
51   1  =  INITIALIZE: PROCEDURE;
52   2  =      CALL MON1(13,0);
53   2  =      END INITIALIZE;
        =
54   1  =  SELECT: PROCEDURE(D);
55   2  =      DECLARE D BYTE;
56   2  =      CALL MON1(14,D);
57   2  =      END SELECT;
        =
58   1  =  OPEN: PROCEDURE(FCB);
59   2  =      DECLARE FCB ADDRESS;
60   2  =      DCNT = MON2(15,FCB);
61   2  =      END OPEN;
        =
```



```
62   1  =  CLOSE: PROCEDURE(FCB);
63   2  =      DECLARE FCB ADDRESS;
64   2  =      DCNT = MON2(16,FCB);
65   2  =      END CLOSE;
        =
66   1  =  SEARCH: PROCEDURE(FCB);
67   2  =      DECLARE FCB ADDRESS;
68   2  =      DCNT = MON2(17,FCB);
69   2  =      END SEARCH;
        =
70   1  =  SEARCHN: PROCEDURE;
71   2  =      DCNT = MON2(18,0);
72   2  =      END SEARCHN;
        =
73   1  =  DELETE: PROCEDURE(FCB);
74   2  =      DECLARE FCB ADDRESS;
75   2  =      CALL MON1(19,FCB);
76   2  =      END DELETE;
        =
77   1  =  DISKREAD: PROCEDURE(FCB) BYTE;
78   2  =      DECLARE FCB ADDRESS;
79   2  =      RETURN MON2(20,FCB);
80   2  =      END DISKREAD;
        =
81   1  =  DISKWRITE: PROCEDURE(FCB) BYTE;
82   2  =      DECLARE FCB ADDRESS;
83   2  =      RETURN MON2(21,FCB);
84   2  =      END DISKWRITE;
        =
85   1  =  MAKE: PROCEDURE(FCB);
86   2  =      DECLARE FCB ADDRESS;
87   2  =      DCNT = MON2(22,FCB);
88   2  =      END MAKE;
        =
89   1  =  RENAME: PROCEDURE(FCB);
90   2  =      DECLARE FCB ADDRESS;
91   2  =      CALL MON1(23,FCB);
92   2  =      END RENAME;
        =
93   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 */
94   1  =  DECLARE (TCBP,CBP) BYTE;   /* TEMP CBP, COMMAND BUFFER POINTER */
        =
95   1  =  READCOM: PROCEDURE;
        =      /* READ INTO COMMAND BUFFER */
96   2  =      MAXLEN = 128;
97   2  =      CALL MON1(10,.MAXLEN);
98   2  =      END READCOM;

99   1     DECLARE MCBP BYTE;

100  1     CONBRK: PROCEDURE BYTE;
           /* CHECK CONSOLE CHARACTER READY */
101  2        RETURN MON2(11,0);
102  2        END CONBRK;

103  1     DECLARE IOBYTE BYTE AT(3H); /* INTEL IOBYTE */
```

```
100   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 */
                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 */
                QUITS  BYTE AT(.CONT(16)),    /* QUIT COPY */
                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 */
                ZEROP  BYTE AT(.CONT(25));    /* ZERO PARITY ON INPUT */

105   1       LIFTHEAD: PROCEDURE;
106   2           CALL MON1(12,0);
107   2         END LIFTHEAD;

108   1       SETDMA: PROCEDURE(A);
109   2           DECLARE A ADDRESS;
110   2           CALL MON1(26,A);
111   2           END SETDMA;

              /* INTELLEC 8 INTEL/ICOM READER INPUT */

112   1     INTIN: PROCEDURE BYTE;
              /* READ THE INTEL / ICOM READER */
113   2         DECLARE PTRI LITERALLY '3',   /* DATA */
                      PTRS LITERALLY '1',   /* STATUS */
                      PTRC LITERALLY '1',   /* COMMAND */

                      PTRG LITERALLY '0CH', /* GO */
                      PTRH LITERALLY '08H'; /* STOP */

              /* STROBE THE READER */
114   2         OUTPUT(PTRC) = PTRG;
115   2         OUTPUT(PTRC) = PTRH;
116   2             DO WHILE NOT ROL(INPUT(PTRS),3); /* NOT READY */
117   3             END;
              /* DATA READY */
118   2         RETURN INPUT(PTRI) AND 7FH;
119   2         END INTIN;


120   1     DECLARE ZEROSUP BYTE,  /* ZERO SUPPRESSION */
              (C3,C2,C1) BYTE;     /* LINE COUNT ON PRINTER  */

121   1       ERROR: PROCEDURE(A);
122   2         DECLARE A ADDRESS, I BYTE;
123   2         CALL PRINT(A); CALL PRINTCHAR(','); CALL PRINTCHAR(' ');
126   2             DO I = TCBP TO CBP;
127   3             IF I < COMLEN THEN CALL PRINTCHAR(COMBUFF(I));
129   3             END;
```



```
130   2         /* ZERO THE COMLEN IN CASE THIS IS A SINGLE COMMAND */
                COMLEN = 0;
                /* DELETE ANY $$$.SUB FILES IN CASE BATCH PROCESSING */
131   2         CALL DELETE(.(0,'$$$     SUB',0));
132   2         CALL CRLF;
133   2         GO TO RETRY;
134   2         END ERROR;

135   1       MOVE: PROCEDURE(S,D,N);
136   2         DECLARE (S,D) ADDRESS, N BYTE;
137   2         DECLARE A BASED S BYTE, B BASED D BYTE;
138   2             DO WHILE (N:=N-1) <> 255;
139   3             B = A; S = S+1; D = D+1;
142   3             END;
143   2         END MOVE;

144   1       FILLSOURCE: PROCEDURE;
              /* FILL THE SOURCE BUFFERS */
145   2         DECLARE (I,J) BYTE;
146   2         NSOURCE = 0;
147   2         CALL SELECT(SDISK);
148   2             DO I = 0 TO NBUF;
                  /* SET DMA ADDRESS TO NEXT BUFFER POSITION */
149   3             CALL SETDMA(.SBUFF(NSOURCE));
150   3             IF (J := DISKREAD(.SOURCE)) <> 0 THEN
151   3                 DO; IF J <> 1 THEN
153   4                     CALL ERROR(.('DISK READ ERROR$'));
                      /* END - OF - FILE */
154   4                 HARDEOF = NSOURCE; /* SET HARD END-OF-FILE */
155   4                 SBUFF(NSOURCE) = ENDFILE; I = NBUF;
157   4                 END; ELSE
158   3             NSOURCE = NSOURCE + 128;
159   3             END;
160   2         NSOURCE = 0;
161   2         END FILLSOURCE;

162   1       WRITEDEST: PROCEDURE;
              /* WRITE OUTPUT BUFFERS UP TO BUT NOT INCLUDING POSITION
              NDEST - THE LOW ORDER 7 BITS OF NDEST ARE ZERO */
163   2         DECLARE (I, J, N) BYTE;
164   2         DECLARE DMA ADDRESS;
165   2         DECLARE (EXTCNT, DATAOK) BYTE;
166   2         IF (N := LOW(SHR(NDEST,7)) - 1) = 255 THEN RETURN ;
168   2         EXTCNT,NDEST = 0;
169   2         CALL SELECT(DDISK);
170   2             DO I = 0 TO N;
                  /* SET DMA ADDRESS TO NEXT BUFFER */
171   3             DMA = .DBUFF(NDEST);
172   3             IF VERIF THEN /* VERIFY MODE */
173   3                 DO;
174   4                 IF DEST(32) = 127 THEN /* END OF EXTENT */
175   4                     DO; CALL MOVE(DMA,80H,80H);
177   5                     DMA = 80H; EXTCNT = EXTCNT + 1;
179   5                     END;
180   4                 END;
181   3             CALL SETDMA(DMA);
182   3             IF DISKWRITE(.DEST) <> 0 THEN
183   3                 CALL ERROR(.('DISK WRITE ERROR$'));
184   3             NDEST = NDEST + 128;
```

```
185   3             END;
186   2         IF VERIF THEN /* VERIFY DATA WRITTEN OK */
187   2             DO; I = N + 1; /* NUMBER OF BUFFERS */
189   3             NDEST = 0;
190   3             IF EXTCNT <> 0 THEN /* WENT OVER CURRENT EXTENT */
191   3                 DO; CALL SETDMA(80H); CALL CLOSE(.DEST);
194   4                 DEST(FREEL) = DEST(FREEL) - EXTCNT;
195   4                 J = DEST(32); /* SAVE CURRENT RECORD NUMBER */
196   4                 CALL OPEN(.DEST);
                        /* ONE OR TWO EXTENTS WHERE WRITTEN */
197   4                 I = I - 128;
                        /* I MAY BE (TWO'S COMPLEMENT) NEGATIVE */
198   4                 DEST(32) = (J - I) AND 7FH;
199   4                 END; ELSE
200   3             DEST(32) = DEST(32) - I; /* NEXT TO READ */
201   3             CALL SETDMA(80H); /* FOR COMPARE */
202   3                 DO I = 0 TO N;
203   4                 DATAOK = DISKREAD(.DEST) = 0;
204   4                 J = 0;
                            /* PERFORM COMPARISON */
205   4                     DO WHILE DATAOK AND J < 80H;
206   5                     DATAOK = BUFFER(J) = DBUFF(NDEST+J);
207   5                     J = J + 1;
208   5                     END;
209   4                 NDEST = NDEST + 128;
210   4                 IF NOT DATAOK THEN
211   4                     CALL ERROR(.('VERIFY ERROR$'));
212   4                 END;
213   3             IF DEST(32) = 128 THEN /* INTO NEXT EXTENT */
214   3                 DO; DEST(FREEL) = DEST(FREEL) + 1;
216   4                 CALL OPEN(.DEST);
217   4                 END;
218   3             END;
219   2         NDEST = 0;
220   2         END WRITEDEST;

221   1     PUTDESTC: PROCEDURE(B);
222   2         DECLARE (B,IOB) BYTE;
                /* WRITE BYTE B TO THE DESTINATION DEVICE GIVEN BY PDEST */
223   2         IF B >= ' ' THEN
224   2             DO; COLUMN = COLUMN + 1;
226   3             IF DELET > 0 THEN /* MAY BE PAST RIGHT SIDE */
227   3                 DO; IF COLUMN > DELET THEN RETURN;
230   4                 END;
231   3             END;
232   2         IOB = IOBYTE; /* IN CASE IT IS ALTERED */
233   2             DO CASE PDEST;
                        /* CASE 0 IS THE DESTINATION FILE */
234   3                 DO;
235   4                 IF NDEST >= DBLEN THEN CALL WRITEDEST;
237   4                 DBUFF(NDEST) = B;
238   4                 NDEST = NDEST+1;
239   4                 END;
                        /* CASE 1 IS ARD (ADDMASTER) */
240   3                 GO TO NOTDEST;
                      /* CASE 2 IS IRD (INTEL/ICOM) */
241   3                 GO TO NOTDEST;
                        /* CASE 3 IS PTR */
242   3                 GO TO NOTDEST;
                        /* CASE 4 IS UR1 */
243   3                 GO TO NOTDEST;
                      /* CASE 5 IS UR2 */
244   3                 GO TO NOTDEST;
                      /* CASE 6 IS RDR */
245   3                 NOTDEST:
                            CALL ERROR(.('NOT A CHARACTER SINK$'));
                      /* CASE 7 IS OUT */
246   3                 CALL OUT(B);
                      /* CASE 8 IS LPT */
247   3                     DO; IOBYTE = 1000$0000B; GO TO LSTL;
250   4                     END;
                      /* CASE 9 IS UL1 */
251   3                     DO; IOBYTE = 1100$0000B; GO TO LSTL;
254   4                     END;
                      /* CASE 10 IS PRN (TABS EXPANDED, LINES LISTED, CHANGED TO LST) */
255   3                     DO; IOBYTE = 1000$0000B; GO TO LSTL;
258   4                     END;
                      /* CASE 11 IS LST */
259   3                 LSTL:
                            CALL MON1(5,B);
                      /* CASE 12 IS PTP */
260   3                 DO; IOBYTE = 0001$0000B; GO TO PUNL;
263   4                 END;
                      /* CASE 13 IS UP1 */
264   3                 DO; IOBYTE = 0010$0000B; GO TO PUNL;
267   4                 END;
                      /* CASE 14 IS UP2 */
268   3                 DO; IOBYTE = 0011$0000B; GO TO PUNL;
271   4                 END;
                      /* CASE 15 IS PUN */
272   3                 PUNL:
                            CALL MON1(4,B);
                        /* CASE 16 IS TTY */
273   3                     DO; IOBYTE = 0; GO TO CONL;
276   4                     END;
                        /* CASE 17 IS CRT */
277   3                     DO; IOBYTE = 1; GO TO CONL;
280   4                     END;
                        /* CASE 18 IS UC1 */
281   3                     DO; IOBYTE = 11B; GO TO CONL;
284   4                     END;
                        /* CASE 19 IS CON */
285   3                     CONL:
                                CALL MON1(2,B);
286   3                 END;
287   2         IOBYTE = IOB;
288   2         END PUTDESTC;

289   1     PRINT1: PROCEDURE(B);
290   2         DECLARE B BYTE;
291   2         IF (ZEROSUP .= ZEROSUP AND B = 0) THEN CALL PUTDESTC(' '); ELSE
293   2             CALL PUTDESTC('0'+B);
294   2         END PRINT1;

295   1     PRINTDIG: PROCEDURE(D);
296   2         DECLARE D BYTE;
297   2         CALL PRINT1(SHR(D,4)); CALL PRINT1(D AND 1111B);
299   2         END PRINTDIG;

300   1     NEWLINE: PROCEDURE;
```


CP/M VERSION
COPYRIGHT © 1976
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA 93950
SER. # _____

```
301   2         ZEROSUP = 1;
302   2         C1 = DEC(C1+ZEROSUP); C2 = DEC(C2 PLUS 0); C3 = DEC(C3 PLUS 0);
305   2         CALL PRINTDIG(C3); CALL PRINTDIG(C2); CALL PRINTDIG(C1);
308   2         CALL PUTDESTC(' ');
309   2         PUTNUM = FALSE;
310   2         END NEWLINE;

311   1     CLEARBUFF: PROCEDURE;
            /* CLEAR OUTPUT BUFFER IN BLOCK MODE TRANSMISION */
312   2         DECLARE NA ADDRESS;
313   2         DECLARE I BYTE;
314   2         I = LOW(NDEST) AND 7FH;   /* REMAINING PARTIAL BUFFER LENGTH */
315   2         NA = NDEST AND 0FF80H;    /* START OF SEGMENT NOT WRITTEN */
316   2         CALL WRITEDEST;           /* CLEARS BUFFERS */
317   2         CALL MOVE(.DBUFF(NA),.DBUFF,I);
            /* DATA MOVED TO BEGINNING OF BUFFER */
318   2         NDEST = I;
319   2         END CLEARBUFF;

320   1     PUTDEST: PROCEDURE(B);
321   2         DECLARE (I,B) BYTE;
            /* WRITE DESTINATION CHARACTER, CHECK TABS AND LINES */
322   2         IF NUMB AND PUTNUM THEN CALL NEWLINE;
324   2         IF BLOCK THEN /* BLOCK MODE TRANSFER */
325   2             DO;
326   3             IF B = XOFF AND PDEST = 0 THEN
327   3                 DO; CALL CLEARBUFF; /* BUFFERS WRITTEN */
329   4                 RETURN; /* DON'T PASS THE X-OFF */
330   4                 END;
331   3             END;
332   2         IF B = TAB THEN /* EXPAND TO NEXT TAB POSITION */
333   2             DO; IF TABS > 0 THEN
335   3                 DO; I = COLUMN;
337   4                 IF NUMB THEN I = I - 7; /* STARTING BIAS ON LINE */
339   4                     DO WHILE I >= TABS;
340   5                     I = I - TABS;
341   5                     END;
342   4                 I = TABS - I;
343   4                     DO WHILE I > 0;
344   5                     I = I - 1; CALL PUTDESTC(' ');
346   5                     END;
347   4                 END;
348   3             ELSE CALL PUTDESTC(B);
349   3             END; ELSE
350   2             DO;
351   3             CALL PUTDESTC(B);
352   3             IF B = CR THEN COLUMN = 0; ELSE
            /* MAY NEED NEWLINE NEXT TIME AROUND */
354   3             PUTNUM = B = LF;
355   3             END;
356   2         END PUTDEST;

357   1     UTRAN: PROCEDURE(B) BYTE;
358   2         DECLARE B BYTE;
            /* TRANSLATE ALPHA TO UPPER CASE */
359   2         IF B >= 110$0001B AND B <= 111$1010B THEN /* LOWER CASE */
360   2             B = B AND 101$1111B; /* TO UPPER CASE */
361   2         RETURN B;
362   2         END UTRAN;
```



```
363   1     LTRAN: PROCEDURE(B) BYTE;
364   2         DECLARE B BYTE;
            /* TRANSLATE TO LOWER CASE ALPHA */
365   2         IF B >= 'A' AND B <= 'Z' THEN B = B OR 10$0000B; /* TO LOWER */
367   2         RETURN B;
368   2         END LTRAN;

369   1     GETSOURCEC: PROCEDURE BYTE;
            /* READ NEXT SOURCE CHARACTER */
370   2         DECLARE (IOB,B,CONCHK) BYTE;

371   2         IF PSOURCE - 1 <= RDR THEN /* 1 ... RDR+1 */
372   2             DO; IF (BLOCK OR NEXT) AND CONBRK THEN
374   3                 DO;
375   4                 IF READCHAR = ENDFILE THEN RETURN ENDFILE;
377   4                 CALL PRINT(.('READER STOPPING',CR,LF,'$'));
378   4                 RETURN XOFF;
379   4                 END;
380   3             END;
381   2         CONCHK = TRUE; /* CONSOLE STATUS CHECK BELOW */
382   2         IOB = IOBYTE; /* SAVE IT IN CASE IT IS ALTERED */
383   2             DO CASE PSOURCE;
            /* CASE 0 IS SOURCE FILE */
384   3                 DO; IF NSOURCE >= SBLEN THEN CALL FILLSOURCE;
387   4                 B = SBUFF(NSOURCE);
388   4                 NSOURCE = NSOURCE + 1;
389   4                 END;
            /* CASE 1 IS INP */
390   3                 B = INP;
            /* CASE 2 IS IRD (INTEL/ICOM) */
391   3                 B = INTIN;
            /* CASE 3 IS PTR */
392   3                     DO; IOBYTE = 0000$0100B; GO TO RDRL;
395   4                     END;
            /* CASE 4 IS UR1 */
396   3                     DO; IOBYTE = 0000$1000B; GO TO RDRL;
399   4                     END;
            /* CASE 5 IS UR2 */
400   3                     DO; IOBYTE = 0000$1100B; GO TO RDRL;
403   4                     END;
            /* CASE 6 IS RDR */
404   3                 RDRL:
                              B = MON2(3,0) AND 7FH;
            /* CASE 7 IS OUT */
405   3                 GO TO NOTSOURCE;
            /* CASE 8 IS LPT */
406   3                 GO TO NOTSOURCE;
            /* CASE 9 IS UL1 */
407   3                 GO TO NOTSOURCE;
            /* CASE 10 IS PRN */
408   3                 GO TO NOTSOURCE;
            /* CASE 11 IS LST */
409   3                 GO TO NOTSOURCE;
            /* CASE 12 IS PTP */
410   3                 GO TO NOTSOURCE;
            /* CASE 13 IS UP1 */
411   3                 GO TO NOTSOURCE;
            /* CASE 14 IS UP2 */
412   3                 GO TO NOTSOURCE;
```

```
                /* CASE 15 IS PUN */
413   3             NOTSOURCE:
                        DO; CALL ERROR(.('NOT A CHARACTER SOURCE$')));
415   4                 END;
                /* CASE 16 IS TTY */
416   3                 DO; IOBYTE = 0; GO TO CONL;
419   4                 END;
                /* CASE 17 IS CRT */
420   3                 DO; IOBYTE = 01B; GO TO CONL;
423   4                 END;
                /* CASE 18 IS UC1 */
424   3                 DO; IOBYTE = 11B; GO TO CONL;
427   4                 END;
                /* CASE 19 IS CON */
428   3             CONL:
                        DO; CONCHK = FALSE; /* DON'T CHECK CONSOLE STATUS */
430   4                 B = MOH2(1,B);
431   4                 END;
432   3            END; /* OF CASES */
433   2        IOBYTE = IOB; /* RESTORE IOBYTE */
434   2        IF ECHO THEN /* COPY TO CONSOLE DEVICE */
435   2            DO; IOB = PDEST; PDEST = CONP; CALL PUTDEST(B);
439   3            PDEST = IOB;
440   3            END;
441   2        IF CONCHK THEN /* TEST FOR CONSOLE CHAR READY */
442   2            DO;
443   3            IF SCOM THEN /* SOURCE IS A COM FILE */
444   3                CONCHK = (CONCNT := CONCNT + 1) = 0; ELSE /* ASCII */
445   3                CONCHK = B = LF;
446   3            IF CONCHK THEN
447   3                DO; IF CONBRK THEN
449   4                    DO;
450   5                    IF READCHAR = ENDFILE THEN RETURN ENDFILE;
452   5                    CALL ERROR(.('ABORTED$'));
453   5                    END;
454   4                END;
455   3            END;
456   2        IF ZEROP THEN B = B AND 7FH;
458   2        IF UPPER THEN RETURN UTRAN(B);
460   2        IF LOWER THEN RETURN LTRAN(B);
462   2        RETURN B;
463   2        END GETSOURCEC;

464   1    GETSOURCE: PROCEDURE BYTE;
                /* GET NEXT SOURCE CHARACTER */
465   2        DECLARE CHAR BYTE;
466   2        MATCH: PROCEDURE(B) BYTE;
                    /* MATCH START AND QUIT STRINGS */
467   3            DECLARE (B,C) BYTE;
468   3            IF (C:=COMBUFF(B:=(B+MATCHLEN))) = ENDFILE THEN /* END MATCH */
469   3                DO; COMBUFF(B) = CHAR; /* SAVE CURRENT CHARACTER */
471   4                RETURN TRUE;
472   4                END;
473   3            IF C = CHAR THEN MATCHLEN = MATCHLEN + 1; ELSE
475   3                MATCHLEN = 0; /* NO MATCH */
476   3            RETURN FALSE;
477   3            END MATCH;
478   2        IF QUITLEN > 0 THEN
479   2            DO; IF (QUITLEN := QUITLEN - 1) = 1 THEN RETURN LF;
482   3            RETURN ENDFILE; /* TERMINATED WITH CR,LF,ENDFILE */
483   3            END;
484   2        DO FOREVER; /* LOOKING FOR START */
485   3        IF FEEDLEN > 0 THEN /* GET SEARCH CHARACTERS */
486   3            DO; FEEDLEN = FEEDLEN - 1;
488   4            CHAR = COMBUFF(FEEDBASE);
489   4            FEEDBASE = FEEDBASE + 1;
490   4            RETURN CHAR;
491   4            END;
492   3        IF (CHAR := GETSOURCEC) = ENDFILE THEN RETURN ENDFILE;
494   3        IF STARTS > 0 THEN /* LOOKING FOR START STRING */
495   3            DO; IF MATCH(STARTS) THEN
497   4                DO; FEEDBASE = STARTS; STARTS = 0;
500   5                FEEDLEN = MATCHLEN + 1;
501   5                END; /* OTHERWISE NO MATCH, SKIP CHARACTER */
502   4            END; ELSE
503   3        IF QUITS > 0 THEN /* PASS CHARACTERS TIL MATCH */
504   3            DO; IF MATCH(QUITS) THEN
506   4                DO; QUITS = 0; QUITLEN = 2;
                        /* SUBSEQUENTLY RETURN CR, LF, ENDFILE */
509   5                RETURN CR;
510   5                END;
511   4            RETURN CHAR;
512   4            END; ELSE
513   3        RETURN CHAR;
514   3        END; /* OF DO FOREVER */
515   2        END GETSOURCE;

516   1    DECLARE DISK BYTE;              /* SELECTED DISK */

517   1      GNC: PROCEDURE BYTE;
518   2        IF (CBP := CBP + 1) >= COMLEN THEN RETURN CR;
520   2        RETURN UTRAN(COMBUFF(CBP));
521   2        END GNC;

522   1      DEBLANK: PROCEDURE;
523   2            DO WHILE (CHAR := GNC) = ' ';
524   3            END;
525   2        END DEBLANK;

526   1      SCAN: PROCEDURE(FCBA);
527   2        DECLARE FCBA ADDRESS,         /* ADDRESS OF FCB TO FILL */
                  FCB BASED FCBA (FSIZE) BYTE,  /* FCB TEMPLATE */
528   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 */

529   2        DELIMITER: PROCEDURE(C) BYTE;
530   3            DECLARE (I,C) BYTE;
531   3            DECLARE DEL(*) BYTE DATA
                    (' =.,:()',CR,LA,LB,RB);
532   3                DO I = 0 TO LAST(DEL);
533   4                IF C = DEL(I) THEN RETURN TRUE;
535   4                END;
536   3            RETURN FALSE;
537   3            END DELIMITER;

538   2        PUTCHAR: PROCEDURE;
539   3            FCB(FLEN:=FLEN+1) = CHAR;
540   3            IF CHAR = WHAT THEN AMBIG = TRUE; /* CONTAINS AMBIGUOUS REF */
```


CP/M VERSION _____
COPYRIGHT © 1976
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA. 93950
SER. # _____

```
542   3           END PUTCHAR;

543   2       FILLQ: PROCEDURE(LEN);
                  /* FILL CURRENT NAME OR TYPE WITH QUESTION MARKS */
544   3           DECLARE LEN BYTE;
545   3           CHAR = WHAT; /* QUESTION MARK */
546   3               DO WHILE FLEN < LEN;
547   4               CALL PUTCHAR;
548   4               END;
549   3           END FILLQ;

550   2       GETFCB: PROCEDURE(I) BYTE;
551   3           DECLARE I BYTE;
552   3           RETURN FCB(I);
553   3           END GETFCB;

554   2       SCANPAR: PROCEDURE;
555   3           DECLARE (I,J) BYTE;
                  /* SCAN OPTIONAL PARAMETERS */
556   3           PARSET = TRUE;
557   3           CHAR = GNC; /* SCAN PAST BRACKET */
558   3               DO WHILE NOT(CHAR = CR OR CHAR = RB);
559   4               IF (I := CHAR - 'A') > 25 THEN /* NOT ALPHA */
560   4                   DO; IF CHAR = ' ' THEN CHAR = GNC; ELSE
563   5                   CALL ERROR(.('BAD PARAMETER$'));
564   5                   END; ELSE
565   4                   DO; /* SCAN PARAMETER VALUE */
566   5                   IF CHAR = 'S' OR CHAR = 'Q' THEN
567   5                       DO; /* START OR QUIT COMMAND */
568   6                       J = CBP + 1; /* START OF STRING */
569   6                           DO WHILE NOT ((CHAR := GNC) = ENDFILE OR CHAR = CR);
570   7                           END;
571   6                        CHAR=GNC;
572   6                       END; ELSE
573   5                   IF (J := (CHAR := GNC) - '0') > 9 THEN J = 1;
                          ELSE
575   5                       DO WHILE (K := (CHAR := GNC) - '0') <= 9;
576   6                       J = J * 10 + K;
577   6                       END;
578   5                   CONT(I) = J;
579   5                   END;
590   4               END;
581   3           CHAR = GNC;
582   3           END SCANPAR;

583   2       CHKSET: PROCEDURE;
584   3           IF CHAR = LA THEN CHAR = '=';
586   3           END CHKSET;

                /* INITIALIZE FILE CONTROL BLOCK TO EMPTY */
587   2       AMBIG = FALSE; TYPE = ERR; CHAR = ' '; FLEN = 0;
591   2           DO WHILE FLEN < FSIZE-1;
592   3           IF FLEN = FNSIZE THEN CHAR = 0;
594   3           CALL PUTCHAR;
595   3           END;

                /* DEBLANK COMMAND BUFFER */
596   2           CALL DEBLANK;

                /* SAVE STARTING POSITION OF SCAN FOR DIAGNOSTICS */
---   -           ----
```



```
597   2           TCBP = CBP;

                  /* MAY BE A SEPARATOR */
598   2           IF DELIMITER(CHAR) THEN
599   2               DO; CALL CHKSET;
601   3               TYPE = SPECL; RETURN;
603   3               END;

                  /* CHECK PERIPHERALS AND DISK FILES */
604   2           DISK = 0;
                  /* CLEAR PARAMETERS */
605   2               DO I = 0 TO 25; CONT(I) = 0;
607   3               END;
608   2           PARSET = FALSE;
609   2           FEEDLEN,MATCHLEN,QUITLEN = 0;
                  /* SCAN NEXT NAME */
610   2               DO FOREVER;
611   3               FLEN = 0;
612   3                   DO WHILE NOT DELIMITER(CHAR);
613   4                   IF FLEN >= NSIZE THEN /* ERROR, FILE NAME TOO LONG */
614   4                       RETURN;
615   4                   IF CHAR = '*' THEN CALL FILLQ(NSIZE); ELSE CALL PUTCHAR;
618   4                   CHAR = GNC;
619   4                   END;

                      /* CHECK FOR DISK NAME OR DEVICE NAME */
620   3                   IF CHAR = ':' THEN
621   3                       DO; IF DISK <> 0 THEN RETURN; /* ALREADY SET */
624   4                       IF FLEN = 1 THEN
                                /* MAY BE DISK NAME A ... Z */
625   4                           DO;
626   5                           IF (DISK := GETFCB(1) - 'A' + 1) > 26 THEN
627   5                           /* ERROR, INVALID DISK NAME */ RETURN;
628   5                           CALL DEBLANK; /* MAY BE DISK NAME ONLY */
629   5                           IF DELIMITER(CHAR) THEN
630   5                               DO; IF CHAR = LB THEN
632   6                                   CALL SCANPAR;
633   6                               CBP = CBP - 1;
634   6                               TYPE = DISKNAME;
635   6                               RETURN;
636   6                               END;
637   5                           END; ELSE

                              /* MAY BE A THREE CHARACTER DEVICE NAME */
638   4                           IF FLEN <> 3 THEN /* ERROR, CANNOT BE DEVICE NAME */
639   4                               RETURN; ELSE

                                    /* LOOK FOR DEVICE NAME */
640   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 */
642   5                               J = 255;
643   5                                   DO K = 0 TO M;
644   6                                   I = 0;
```

```
645  6                          DO WHILE ((I:=I+1) <= 3) AND
                                IO(J+I) = GETFCB(I);
646  7                          END;
647  6                      IF I = 4 THEN /* COMPLETE MATCH */
648  6                          DO; TYPE = PERIPH;
                                /* SCAN PARAMETERS */
650  7                          IF GNC = LB THEN CALL SCANPAR;
652  7                          CBP = CBP - 1; CHAR = K;
654  7                          RETURN;
655  7                          END;
656  6                      /* OTHERWISE TRY NEXT DEVICE */ J = J + 3;
657  6                      END;

658  5                  /* ERROR, NO DEVICE NAME MATCH */ RETURN;
659  5                  END;
660  4              IF CHAR = LB THEN /* PARAMETERS FOLLOW */
661  4                  CALL SCANPAR;
662  4              END; ELSE

             /* CHAR IS NOT '.', SO FILE NAME IS SET. SCAN REMAINDER */
663  3              DO; IF FLEN = 0 THEN /* ERROR, NO PRIMARY NAME */
665  4                  RETURN;
666  4              FLEN = FNAM;
667  4              IF CHAR = '.' THEN /* SCAN FILE TYPE */
668  4                  DO WHILE NOT DELIMITER(CHAR := GNC);
669  5                  IF FLEN >= FNSIZE THEN
670  5                  /* ERROR, TYPE FIELD TOO LONG */ RETURN;
671  5                  IF CHAR = '*' THEN CALL FILLQ(FNSIZE);
673  5                      ELSE CALL PUTCHAR;
674  5                  END;

675  4              IF CHAR = LB THEN
676  4                  CALL SCANPAR;
               /* RESCAN DELIMITER NEXT TIME AROUND */
677  4              CBP = CBP - 1;
678  4              TYPE = FILE;
               /* DISK IS THE SELECTED DISK (1 2 3 ... ) */
679  4              IF DISK = 0 THEN DISK = CDISK + 1; /* DEFAULT */
681  4              FCB(0),FCB(32) = 0;
682  4              RETURN;
683  4              END;
684  3          END;
685  2      END SCAN;

686  1      NULLS: PROCEDURE;
           /* SEND 40 NULLS TO OUTPUT DEVICE */
687  2          DECLARE I BYTE;
688  2              DO I = 0 TO 39; CALL PUTDEST(0);
690  3              END;
691  2      END NULLS;


692  1      DECLARE FEXTH(FEXTL) BYTE,      /* HOLDS DESTINATION FILE TYPE */
           COPYING BYTE;                   /* TRUE WHILE COPYING TO DEST FILE */

693  1      MOVEXT: PROCEDURE(A);
694  2          DECLARE A ADDRESS;
           /* MOVE THREE CHARACTER EXTENT INTO DEST FCB */
695  2          CALL MOVE(A,.DEST(FEXT),FEXTL);
696  2          END MOVEXT;
```



```
697  1      EQUAL: PROCEDURE(A,B) BYTE;
               /* COMPARE THE STRINGS AT A AND B UNTIL EITHER A MISMATCH OR
               A '$' IS ENCOUNTERED IN STRING B */
698  2          DECLARE (A,B) ADDRESS,
               (SA BASED A, SB BASED B) BYTE;
699  2              DO WHILE SB <> '$';
700  3              IF SB <> SA THEN RETURN FALSE;
702  3              A = A + 1; B = B + 1;
704  3              END;
705  2          RETURN TRUE;
706  2          END EQUAL;

707  1      READ$EOF: PROCEDURE BYTE;
               /* RETURN TRUE IF END OF FILE */
708  2          CHAR = GETSOURCE;
709  2          IF SCOM THEN RETURN HARDEOF (= NSOURCE;
711  2          RETURN CHAR = ENDFILE;
712  2          END READ$EOF;


713  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       */


714  2          DECLARE XOFFSET BYTE; /* TRUE IF XOFF RECVD */
715  2          DECLARE NOERRS BYTE; /* TRUE IF NO ERRORS IN THIS RECORD */

716  2          PRINTERR: PROCEDURE(A);
               /* PRINT ERROR MESSAGE IF NOERRS TRUE */
717  3              DECLARE A ADDRESS;
718  3              IF NOERRS THEN
719  3                  DO; NOERRS = FALSE;
721  4                  CALL PRINT(A);
722  4                  END;
723  3              END PRINTERR;

724  2          CHECKXOFF: PROCEDURE;
725  3              IF XOFFSET THEN
726  3                  DO; XOFFSET = FALSE;
728  4                  CALL CLEARBUFF;
729  4                  END;
730  3              END CHECKXOFF;

731  2          SAVECHAR: PROCEDURE BYTE;
               /* READ CHARACTER AND SAVE IN BUFFER */
732  3              DECLARE I BYTE;
733  3              IF NOERRS THEN
734  3                  DO;
735  4                      DO WHILE (I := GETSOURCE) = XOFF; XOFFSET = TRUE;
737  5                      END;
738  4                  HBUFF(HSOURCE) = I;
739  4                  IF (HSOURCE := NSOURCE + 1) >= LAST(HBUFF) THEN
740  4                      CALL PRINTERR(.('RECORD TOO LONG$'));
741  4                  RETURN I;
742  4                  END;
743  3              RETURN ENDFILE; /* ON ERROR FLAG */
```

```
744   3          END SAVECHAR;

745   2      DECLARE (M, RL, CS, RT) BYTE,
                  LDA ADDRESS;  /* LOAD ADDRESS WHICH FOLLOWS : */

746   2      READHEX: PROCEDURE BYTE;
747   3          DECLARE H BYTE;
748   3          IF (H := SAVECHAR) - '0' <= 9 THEN RETURN H-'0';
750   3          IF H - 'A' > 5 THEN
751   3              CALL PRINTERR(.('INVALID DIGIT$'));
752   3          RETURN H - 'A' + 10;
753   3          END READHEX;

754   2      READBYTE: PROCEDURE BYTE;
                  /* READ TWO HEX DIGITS */
755   3          RETURN SHL(READHEX,4) OR READHEX;
756   3          END READBYTE;

757   2      READCS: PROCEDURE BYTE;
                  /* READ BYTE WITH CHECKSUM */
758   3          RETURN CS := CS + READBYTE;
759   3          END READCS;

760   2      READADDR: PROCEDURE ADDRESS;
                  /* READ DOUBLE BYTE WITH CHECKSUM */
761   3          RETURN SHL(DOUBLE(READCS),8) OR READCS;
762   3          END READADDR;

763   2      NOERRS = TRUE; /* NO ERRORS DETECTED IN THIS RECORD */

              /* READ NEXT RECORD */
                  /* SCAN FOR THE ',' */
764   2          HSOURCE = 0;
765   2              DO WHILE (CS := SAVECHAR) <> ',';
766   3              HSOURCE = 0;
767   3              IF CS = ENDFILE THEN
768   3                  DO; CALL PRINT(.('END OF FILE, CTL-Z',WHAT,'$'));
770   4                  IF READCHAR = ENDFILE THEN RETURN 1;
772   4                      ELSE HSOURCE = 0;
773   4                  END;
774   3              CALL CHECKKOFF;
775   3              END;

                  /* ',' FOUND */
776   2          CS = 0;
777   2          IF (RL := READCS) = 0 THEN /* END OF TAPE */
778   2              DO; DO WHILE (RL := SAVECHAR) <> ENDFILE;
780   4                  CALL CHECKXOFF;
781   4                  END;
782   3              IF NOERRS THEN RETURN 1;
784   3              RETURN 2;
785   3              END;

              /* RECORD LENGTH IS NOT ZERO */
786   2          LDA = READADDR; /* LOAD ADDRESS */

              /* READ WORDS UNTIL RECORD LENGTH EXHAUSTED */
787   2          RT = READCS; /* RECORD TYPE */
788   2             DO WHILE RL <> 0 AND NOERRS; RL = RL - 1;
790   3             M = READCS;
```



```
                  /* INCREMENT LA HERE FOR EXACT ADDRESS */
791   3             END;

              /* CHECK SUM */
792   2          IF CS + READBYTE <> 0 THEN
793   2              CALL PRINTERR(.('CHECKSUM ERROR$'));

794   2          CALL CHECKXOFF;
795   2          IF NOERRS THEN RETURN 0;
797   2          RETURN 2;
798   2          END HEXRECORD;

799   1      READTAPE: PROCEDURE;
                  /* READ HEX FILE FROM HIGH SPEED READER TO 'HEX' FILE,
                  CHECK EACH RECORD FOR VALID DIGITS, AND PROPER CHECKSUM */
800   2          DECLARE (I,A) BYTE;
801   2              DO FOREVER;
802   3                  DO WHILE (I := HEXRECORD) <= 1;
803   4                  IF NOT (I = 1 AND IGNOR) THEN
804   4                      DO A = 1 TO HSOURCE;
805   5                      CALL PUTDEST(HBUFF(A-1));
806   5                      END;
807   4                  CALL PUTDEST(CR); CALL PUTDEST(LF);
809   4                  IF I = 1 THEN /* END OF TAPE ENCOUNTERED */
810   4                      RETURN;
811   4                  END;
812   3              CALL CRLF; HBUFF(HSOURCE) = '$';
814   3              CALL PRINT(.HBUFF);
815   3              CALL PRINT(.('CORRECT ERROR, TYPE RETURN OR CTL-Z$'));
816   3              CALL CRLF;
817   3              IF READCHAR = ENDFILE THEN RETURN;
819   3              END;
820   2          END READTAPE;

821   1      FORMERR: PROCEDURE;
822   2          CALL ERROR(.('INVALID FORMAT$'));
823   2          END FORMERR;

824   1      SETUPDEST: PROCEDURE;
825   2          CALL SELECT(DDISK);
826   2          DHEX = EQUAL(.DEST(FEXT),.('HEX$'));
827   2          CALL MOVE(.DEST(FEXT),.FEXTH,FEXTL); /* SAVE TYPE */
828   2          CALL MOVEXT(.('$$$')); CALL DELETE(.DEST); /* REMOVE OLD $$$ FILE */
830   2          CALL MAKE(.DEST); /* CREATE A NEW ONE */
831   2          IF DCNT = 255 THEN CALL ERROR(.('NO DIRECTORY SPACE$'));
833   2          DEST(32),NDEST = 0;
834   2          END SETUPDEST;

835   1      SETUPSOURCE: PROCEDURE;
836   2          HARDEOF = 0FFFFH;
837   2          CALL SELECT(SDISK);
838   2          CALL OPEN(.SOURCE);
839   2          IF DCNT = 255 THEN CALL ERROR(.('NO FILE$'));
841   2          SOURCE(32) = 0;
                  /* CAUSE IMMEDIATE READ */
842   2          SCOM = EQUAL(.SOURCE(FEXT),.('COM$')) OR
                         EQUAL(.SOURCE(FEXT),.('CMI$'));
843   2          NSOURCE = SBLEN;
844   2          END SETUPSOURCE;
```

```
845   1     CHECK$STRINGS: PROCEDURE;
846   2         IF STARTS > 0 THEN
847   2             CALL ERROR(.('START NOT FOUND$'));
848   2         IF QUITS  > 0 THEN
849   2             CALL ERROR(.('QUIT NOT FOUND$'));
850   2         END CHECK$STRINGS;

851   1     CLOSEDEST: PROCEDURE;
              /* CLEAR BUFFER */
852   2         DO WHILE (LOW(NDEST) AND 7FH) <> 0;
853   3             CALL PUTDEST(ENDFILE);
854   3             END;
855   2         CALL CHECK$STRINGS;
856   2         CALL WRITEDEST;
857   2         CALL SELECT(DDISK);
858   2         CALL CLOSE(.DEST);
859   2         IF DCNT = 255 THEN CALL ERROR(.('WRITE PROTECTED',WHAT,'$'));
861   2         CALL MOVEXT(.FEXTH); /* RECALL ORIGINAL TYPTE */
862   2         CALL DELETE(.DEST); /* REMOVE OLD FILE */
863   2         CALL MOVE(.DEST,.DEST(16),16); /* READY FOR RENAME */
864   2         CALL MOVEXT(.('$$$'));
865   2         CALL RENAME(.DEST);
866   2         END CLOSEDEST;

867   1     SIZE$NBUF: PROCEDURE;
              /* COMPUTE NUMBER OF BUFFERS - 1 FROM DBLEN */
868   2         NBUF = (SHR(DBLEN,7) AND 0FFH) - 1;
              /* COMPUTED AS DBLEN/128-1, WHERE DBLEN <= 32K (AND THUS
              NBUF RESULTS IN A VALUE (= 2**15/2**7-1 = 2**8-1 = 255) */
869   2         END SIZE$NBUF;

870   1     SET$DBLEN: PROCEDURE;
              /* ABSORB THE SOURCE BUFFER INTO THE DEST BUFFER */
871   2         SBASE = .MEMORY;
872   2         IF DBLEN >= 4800H THEN DBLEN = 7F80H; ELSE
874   2             DBLEN = DBLEN + SBLEN;
875   2         CALL SIZE$NBUF;
876   2         END SET$DBLEN;

877   1     SIZE$MEMORY: PROCEDURE;
              /* SET UP SOURCE AND DESTINATION BUFFERS */
878   2         SBASE = .MEMORY + SHR(MEMSIZE - .MEMORY,1);
879   2         SBLEN, DBLEN = SHR((MEMSIZE - .MEMORY) AND 0FF80H,1);
880   2         CALL SIZE$NBUF;
881   2         END SIZE$MEMORY;

882   1     COPYCHAR: PROCEDURE;
              /* PERFORM THE ACTUAL COPY FUNCTION */
883   2         DECLARE RESIZED BYTE; /* TRUE IF SBUFF AND DBUFF COMBINED */
884   2         IF (RESIZED := (BLOCK AND PSOURCE <> 0)) THEN /* BLOCK MODE */
885   2             CALL SET$DBLEN; /* ABSORB SOURCE BUFFER */
886   2         IF NEXT OR IGNOR THEN /* HEX FILE */
887   2             CALL READTAPE; ELSE
888   2             DO WHILE NOT READ$EOF;
889   3             CALL PUTDEST(CHAR);
890   3             END;
891   2         IF RESIZED THEN
892   2             DO; CALL CLEARBUFF;
894   3             CALL SIZE$MEMORY;
895   3             END;
896   2         END COPYCHAR;

897   1     SIMPLECOPY: PROCEDURE;
898   2         DECLARE (FASTCOPY,I) BYTE;
899   2         REAL$EOF: PROCEDURE BYTE;
900   3             RETURN HARDEOF <> 0FFFFH;
901   3             END REALEOF;
902   2         CALL SIZE$MEMORY;
903   2         TCBP = MCBP; /* FOR ERROR TRACING */
904   2         CALL SETUPDEST;
905   2         CALL SETUPSOURCE;
              /* FILES READY FOR DIRECT COPY */
906   2         FASTCOPY = TRUE;
                  /* LOOK FOR PARAMETERS */
907   2             DO I = 0 TO 25;
908   3             IF CONT(I) <> 0 THEN
909   3                 DO;
910   4                 IF NOT(I = 14 OR I = 21) THEN
                          /* NOT OBJ OR VERIFY */
911   4                 FASTCOPY = FALSE;
912   4                 END;
913   3             END;
914   2         IF FASTCOPY THEN /* COPY DIRECTLY TO DBUFF */
915   2             DO; CALL SET$DBLEN; /* EXTEND DBUFF */
917   3                 DO WHILE NOT REAL$EOF;
918   4                 CALL FILLSOURCE;
919   4                 IF REAL$EOF THEN
920   4                     NDEST = HARDEOF; ELSE NDEST = DBLEN;
922   4                 CALL WRITEDEST;
923   4                 END;
924   3             END; ELSE
925   2         CALL COPYCHAR;
926   2         CALL CLOSEDEST;
927   2         END SIMPLECOPY;

928   1     MULTCOPY: PROCEDURE;
929   2         DECLARE (NEXTDIR, NCOPIED) BYTE;
930   2         PRNAME: PROCEDURE;
                  /* PRINT CURRENT FILE NAME */
931   3             DECLARE (I,C) BYTE;
932   3             CALL CRLF;
933   3                 DO I = 1 TO FNSIZE;
934   4                 IF (C := DEST(I)) <> ' ' THEN
935   4                       DO; IF I = FEXT THEN CALL PRINTCHAR('.');
936   5                       CALL PRINTCHAR(C);
939   5                       END;
940   4                 END;
941   3             END PRNAME;

942   2         NEXTDIR,NCOPIED = 0;
943   2             DO FOREVER;
                  /* FIND A MATCHING ENTRY */
944   3             CALL SELECT(SDISK);
945   3             CALL SETDMA(.BUFFER);
946   3             CALL SEARCH(.SEARFCB);
947   3                 DO WHILE DCNT < NEXTDIR;
948   4                 CALL SEARCHN;
949   4                 END;
                  /* FILE CONTROL BLOCK IN BUFFER */
950   3             IF DCNT = 255 THEN
```


CP/M VERSION
COPYRIGHT © 1976
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA. 93950
SER. #

```
951   3                 DO; IF NCOPIED = 0 THEN
953   4                 CALL ERROR(.('NOT FOUND$')); CALL CRLF;
955   4                 RETURN;
956   4                 END;
957   3             NEXTDIR = DCNT + 1;
                    /* GET THE FILE CONTROL BLOCK NAME TO DEST */
958   3             CALL MOVE(.BUFFER+SHL(DCNT AND 11B,5),.DEST,16);
959   3             CALL MOVE(.DEST,.SOURCE,16); /* FILL BOTH FCB'S */
968   3             IF (NCOPIED := NCOPIED + 1) = 1 THEN
961   3                 CALL PRINT(.('COPYING -$'));
962   3             CALL PRNAME;
963   3             CALL SIMPLECOPY;
964   3             END;
965   2         END MULTCOPY;

966   1     SET$SDISK: PROCEDURE;
967   2         IF DISK > 0 THEN SDISK = DISK - 1; ELSE SDISK = CDISK;
970   2         END SET$SDISK;

971   1     SET$DDISK: PROCEDURE;
972   2         IF PARSET THEN /* PARAMETERS PRESENT */ CALL FORMERR;
974   2         IF DISK > 0 THEN DDISK = DISK - 1; ELSE DDISK = CDISK;
977   2         END SET$DDISK;

978   1     CHECK$DISK: PROCEDURE;
979   2         IF DDISK = SDISK THEN CALL FORMERR;
981   2         END CHECK$DISK;

982   1     CHECK$EOL: PROCEDURE;
983   2         CALL DEBLANK;
984   2         IF CHAR <> CR THEN CALL FORMERR;
986   2         END CHECK$EOL;

987   1     SCANDEST: PROCEDURE(COPYFCB);
988   2         DECLARE COPYFCB ADDRESS;
989   2         CALL SET$SDISK;
990   2         CALL CHECK$EOL;
991   2         CALL MOVE(.SOURCE,COPYFCB,33);
992   2         CALL CHECK$DISK;
993   2         END SCANDEST;

994   1     SCANEQL: PROCEDURE;
995   2         CALL SCAN(.SOURCE);
996   2         IF NOT (TYPE = SPECL AND CHAR = '=') THEN CALL FORMERR;
998   2         MCBP = CBP; /* FOR ERROR PRINTING */
999   2         END SCANEQL;


                  /* BUFFER AT 80H CONTAINS REMAINDER OF LINE TYPED
                  FOLLOWING THE COMMAND 'PIP' - IF ZERO THEN PROMPT TIL CR */
1000  1           CALL MOVE(80H,.COMLEN,80H);
1001  1           MULTCOM = COMLEN = 0;

                  /* GET CURRENT DISK */
1002  1           CDISK = MOD2(25,0);

1003  1           RETRY:
                  /* ENTER HERE ON ERROR EXIT FROM THE PROCEDURE 'ERROR' */
                    CALL SIZE$MEMORY;
                    /* MAIN PROCESSING LOOP.  PROCESS UNTIL CR ONLY */
....  .           ..  ....  ..

1004  1           DO FOREVER;
1005  2           CALL LIFTHEAD;
1006  2           CONCNT,COLUMN = 0; /* PRINTER TABS */
                  /* READ FROM CONSOLE IF NOT A ONELINER */
1007  2           IF MULTCOM THEN
1008  2               DO; CALL PRINTCHAR('*'); CALL READCOM;
1011  3               CALL CRLF;
1012  3               END;
1013  2           CBP = 255;
1014  2           IF COMLEN = 0 THEN /* SINGLE CARRIAGE RETURN */
1015  2               DO; CALL SELECT(CDISK);
1017  3               CALL EXIT;
1018  3               END;

                  /* LOOK FOR SPECIAL CASES FIRST */
1019  2           DDISK,SDISK,PSOURCE,PDEST = 0;
1020  2           CALL SCAN(.DEST);
1021  2           IF TYPE = PERIPH THEN GO TO SIMPLECOM;
1023  2           IF TYPE = DISKNAME THEN
1024  2               DO; DDISK = DISK - 1;
1026  3               CALL SCANEQL;
1027  3               CALL SCAN(.SOURCE);
                      /* MAY BE MULTI COPY */
1028  3               IF TYPE <> FILE THEN CALL FORMERR;
1030  3               IF AMBIG THEN
1031  3                   DO; CALL SCANDEST(.SEARFCB);
1033  4                   CALL MULTCOPY;
1034  4                   END; ELSE
1035  3                   DO; CALL SCANDEST(.DEST);
                          /* FORM IS A:=B:UFN */
1037  4                   CALL SIMPLECOPY;
1038  4                   END;
1039  3               GO TO ENDCOM;
1040  3               END;


1041  2           IF TYPE <> FILE OR AMBIG THEN CALL FORMERR;
1043  2             CALL SET$DDISK;
1044  2             CALL SCANEQL;
1045  2             CALL SCAN(.SOURCE);
1046  2             IF TYPE = DISKNAME THEN
1047  2                 DO;
1048  3                 CALL SET$SDISK; CALL CHECK$DISK;
1050  3                 CALL MOVE(.DEST,.SOURCE,33);
1051  3                 CALL CHECK$EOL;
1052  3                 CALL SIMPLECOPY;
1053  3                 GO TO ENDCOM;
1054  3                 END;
                    /* MAY BE POSSIBLE TO DO A FAST DISK COPY */
1055  2             IF TYPE = FILE THEN /* FILE TO FILE */
1056  2                 DO; CALL DEBLANK; IF CHAR <> CR THEN GO TO SIMPLECOM;
                        /* FILE TO FILE */
1060  3                 CALL SET$SDISK;
1061  3                 CALL SIMPLECOPY;
1062  3                 GO TO ENDCOM;
1063  3                 END;

1064  2       SIMPLECOM:
                    CBP = 255; /* READY FOR RESCAN */

                    ..  ....  ..  ....  ....  ....
```


CP/M VERSION
COPYRIGHT © 1976
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA. 93950
SER. #

```
                /* OTHERWISE PROCESS SIMPLE REQUEST */
1065    2       CALL SCAN(.DEST);
1066    2       IF (TYPE < FILE) OR AMBIG THEN /* DELIMITER OR ERROR */
1067    2           CALL ERROR(.('UNRECOGNIZED DESTINATION$'));

1068    2       DHEX = FALSE;
1069    2       IF TYPE = FILE THEN
1070    2           DO; /* DESTINATION IS A FILE, SAVE EXTENT NAME */
1071    3           CALL SET$DDISK;
1072    3           CALL SETUPDEST;
1073    3           CHAR = 255;
1074    3           END; ELSE
                /* PERIPHERAL NAME */
1075    2       IF CHAR >= NULP OR CHAR <= RDR THEN CALL ERROR(.('CANNOT WRITE$'));

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

                /* NOW SCAN THE DELIMITER */
1079    2       CALL SCAN(.SOURCE);
1080    2       IF TYPE <> SPECL OR CHAR <> '=' THEN
1081    2           CALL ERROR(.('INVALID PIP FORMAT$'));

                /* OTHERWISE SCAN AND COPY UNTIL CR */
1082    2       C1,C2,C3 = 0; /* CLEAR LINE COUNTERS */
1083    2       COPYING = TRUE;
1084    2           DO WHILE COPYING;
1085    3           CALL SCAN(.SOURCE);
1086    3           SCOM = FALSE;
1087    3           IF TYPE = FILE AND NOT AMBIG THEN /* A SOURCE FILE */
1088    3               DO;
1089    4               CALL SET$SDISK;
1090    4               CALL SETUPSOURCE;
1091    4               CHAR = 255;
1092    4               END; ELSE

1093    3           IF TYPE <> PERIPH OR (CHAR <= LST AND CHAR > RDR) THEN
1094    3               CALL ERROR(.('CANNOT READ$'));


              COLUMN = 0;
1096    3         PUTNUM = TRUE; /* CAUSES IMMEDIATE NEWLINE IF NUMB SET */
1097    3         SCOM = SCOM OR OBJ; /* MAY BE ABSOLUTE COPY */
1098    3         PSOURCE = CHAR + 1;
1099    3         IF CHAR = NULP THEN CALL NULLS; ELSE
1101    3         IF CHAR = EOFP THEN CALL PUTDEST(ENDFILE); ELSE
1103    3             DO; /* DISK COPY */
1104    4             IF (CHAR < HSRDR AND DHEX) THEN HEXT = 1;
                      /* HEX FILE SET IF SOURCE IS RDR AND DEST IS HEX FILE */
1106    4             IF PDEST = PRNT THEN
1107    4                 DO; TABS = 0; NUMB = 1;
1108    5                 END;
1111    4             CALL COPYCHAR;
1112    4             END;

1113    3         CALL CHECK$STRINGS;
                  /* READ ENDFILE, GO TO NEXT SOURCE */
1114    3         CALL SCAN(.SOURCE);
1115    3         IF TYPE <> SPECL OR (CHAR <> ',' AND CHAR <> CR) THEN
1116    3             CALL ERROR(.('INVALID SEPARATOR$'));

----    -             ...
```



```
1117    3             COPYING = CHAR <> CR;
1118    3             END;

                  /* IF NECESSARY, CLOSE FILE OR PUNCH TRAILER */
1119    2         IF PDEST = PUNP THEN
1120    2             DO; CALL PUTDEST(ENDFILE); CALL NULLS;
1123    3             END;
1124    2         IF PDEST = 0 THEN /* FILE HAS TO BE CLOSED AND RENAMED */
1125    2             CALL CLOSEDEST;

                  /* COMLEN SET TO 0 IF NOT PROCESSING MULTIPLE COMMANDS */
1126    2     ENDCOM:
                  COMLEN = MULTCOM;

1127    2         END; /* DO FOREVER */
1128    1     END;
```



```
MODULE INFORMATION:

      CODE AREA SIZE     = 1947H    6471D
      VARIABLE AREA SIZE = 01CAH     458D
      MAXIMUM STACK SIZE = 0032H      50D
      1396 LINES READ
      0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
1D4FH SYM  MEMORY
0200H SYM  COPYRIGHT
0236H SYM  INPLOC
0238H SYM  OUTLOC
0736H SYM  OUT
1855H SYM  B
0743H SYM  INP
074EH SYM  TIMEOUT
1886H SYM  EXIT
1E53H SYM  COLUMN
1B99H SYM  AMBIG
1B9AH SYM  PARSET
1B9BH SYM  FEEDBASE
1B9CH SYM  FEEDLEN
1B9DH SYM  MATCHLEN
1B9EH SYM  QUITLEN
1B9FH SYM  NBUF
1B90H SYM  CDISK
0B80H SYM  BUFFER
005CH SYM  SFAPFCB
0006H SYM  MEMSIZE
1891H SYM  SBLEN
1893H SYM  DBLEH
1895H SYM  SBASE
1D4FH SYM  DBUFF
1897H SYM  SDISK
1898H SYM  SCOM
1899H SYM  DHEX
189AH SYM  SOURCE
1E6BH SYM  DEST
18DCH SYM  DDISK
18DDH SYM  HBUFF
1C2DH SYM  HSOURCE
1C2EH SYM  HSOURCE
1C30H SYM  HARDEOF
1C32H SYM  HDEST
1C34H SYM  PDEST
1C35H SYM  PSOURCE
1C36H SYM  MULTCOM
1C37H SYM  PUTHUM
1C38H SYM  CONCHT
1C39H SYM  CHAR
1C3AH SYM  TYPE
1C3BH SYM  FLEN
0754H SYM  MON1
1C3CH SYM  F
1C3DH SYM  A
075CH SYM  L1
0760H SYM  MON2
1C3FH SYM  F
1C40H SYM  A
0768H SYM  L2
076EH SYM  READRDP
0777H SYM  READCHAR
0780H SYM  PRINTCHAR
1C42H SYM  CHAR
0790H SYM  CRLF
079BH SYM  PRINT
1C43H SYM  A
1C45H SYM  DCNT
07AEH SYM  INITIALIZE
07E7H SYM  SELECT
1C46H SYM  D
07C7H SYM  OPEN
1C47H SYM  FCB
07DAH SYM  CLOSE
1C49H SYM  FCB
07FDH SYM  SEARCH
1C4BH SYM  FCB
0808H SYM  SEARCHN
080CH SYM  DELETE
1C4DH SYM  FCB
081CH SYM  DISKREAD
1C4FH SYM  FCB
082CH SYM  DISKWRITE
1C51H SYM  FCB
083CH SYM  MAKE
1C53H SYM  FCB
084FH SYM  RENAME
1C55H SYM  FCB
1C57H SYM  CBUFF
1C57H SYM  MAXLEN
1C58H SYM  COMLEN
1C59H SYM  COMBUFF
1CD9H SYM  TCBP
1CDAH SYM  CBP
085FH SYM  READCOM
1CDBH SYM  MCBP
086DH SYM  CONBRK
0003H SYM  IOBYTE
1CDCH SYM  CONT
1CDDH SYM  BLOCK
1CDFH SYM  DELET
1CE0H SYM  ECHO
1CE3H SYM  HEXT
1CE4H SYM  IGNOR
1CE7H SYM  LOWER
1CE9H SYM  NUMB
1CEAH SYM  OBJ
1CECH SYM  QUITS
1CEEH SYM  STARTS
1CEFH SYM  TABS
1CF0H SYM  UPPER
1CF1H SYM  VERIF
1CF5H SYM  ZEROP
0876H SYM  LIFTHEAD
087FH SYM  SETDMA
1CF6H SYM  A
088FH SYM  INTIN
1CF8H SYM  ZEROSUP
1CF9H SYM  C3
1CFAH SYM  C2
1CFBH SYM  C1
08A9H SYM  ERROR
1CFCH SYM  A
1CFEH SYM  I
0901H SYM  MOVE
1CFFH SYM  S
1D01H SYM  D
1D03H SYM  N
0938H SYM  FILLSOURCE
1D04H SYM  I
1D05H SYM  J
09ABH SYM  WRITEDEST
1D06H SYM  I
1D07H SYM  J
1D08H SYM  N
1D09H SYM  DMA
1D0BH SYM  EXTCNT
1D0CH SYM  DATAOK
0B2BH SYM  PUTDESTC
1D0DH SYM  B
1D0EH SYM  IOB
0B98H SYM  NOTDEST
0BCCH SYM  LSTL
0BFBH SYM  PUNL
0C2AH SYM  CONL
0C67H SYM  PRINT1
1DAFH SYM  B
0C8EH SYM  PRINTDIG
1D10H SYM  D
0CA9H SYM  NEWLINE
0CE5H SYM  CLEARBUFF
1D11H SYM  NA
1D13H SYM  I
0D12H SYM  PUTDEST
1D14H SYM  B
1D15H SYM  I
0DD1H SYM  UTRAN
1D16H SYM  B
0DF8H SYM  LTRAN
1D17H SYM  B
0E1FH SYM  GETSOURCEC
1D18H SYM  IOB
1D19H SYM  B
1D1AH SYM  CONCHK
0EC3H SYM  RDRL
0EEBH SYM  NOTSOURCE
0F15H SYM  CONL
0FF0H SYM  GETSOURCE
1D1BH SYM  CHAR
1093H SYM  MATCH
1D1CH SYM  B
1D1DH SYM  C
1D1EH SYM  DISK
10D8H SYM  GNC
10F7H SYM  DEBLANK
1106H SYM  SCAN
1D1FH SYM  FCBA
1D21H SYM  I
1D22H SYM  J
1D23H SYM  K
1320H SYM  DELIMITER
1D24H SYM  C
1D25H SYM  I
023AH SYM  DEL
134FH SYM  PUTCHAR
136FH SYM  FILLQ
1D26H SYM  LEN
1389H SYM  GETFCB
1D27H SYM  I
1399H SYM  SCANPAR
1D28H SYM  I
1D29H SYM  J
1488H SYM  CHKSET
1D2AH SYM  I
1D2BH SYM  J
1D2CH SYM  K
0245H SYM  IO
1496H SYM  NULLS
1D2DH SYM  I
1D2EH SYM  FEXTH
1D31H SYM  COPYING
14B1H SYM  MOVEXT
1D32H SYM  A
14C4H SYM  EQUAL
1D34H SYM  A
1D36H SYM  B
14F8H SYM  READEOF
151DH SYM  HEXRECORD
1D38H SYM  XOFFSET
1D39H SYM  NOERRS
15D4H SYM  PRINTERR
1D3AH SYM  A
15EFH SYM  CHECKXOFF
15FFH SYM  SAVECHAR
1D3CH SYM  I
1D3DH SYM  M
1D3EH SYM  RL
1D3FH SYM  CS
1D40H SYM  RT
1D41H SYM  LDA
163FH SYM  READHEX
1D43H SYM  H
166EH SYM  READBYTE
167DH SYM  READCS
1686H SYM  READADDR
169CH SYM  READTAPE
1D44H SYM  I
1D45H SYM  A
171FH SYM  FORMERR
1726H SYM  SETUPDEST
1771H SYM  SETUPSOURCE
17B7H SYM  CHECKSTRINGS
17D6H SYM  CLOSEDEST
182FH SYM  SIZEHBUF
1843H SYM  SETDBLEN
186DH SYM  SIZEMEMORY
189DH SYM  COPYCHAR
1D46H SYM  RESIZED
18E4H SYM  SIMPLECOPY
1D47H SYM  FASTCOPY
1D48H SYM  I
1973H SYM  REALEOF
1981H SYM  MULTCOPY
1D49H SYM  NEXTDIR
1D4AH SYM  NCOPIED
1A0FH SYM  PRNAME
1D48H SYM  I
1A4EH SYM  SETSDISK
1A68H SYM  SETDDISK
1A8CH SYM  CHECKDISK
1A9AH SYM  CHECKEOL
1AA9H SYM  SCANDEST
1D4DH SYM  COPYFCB
1AC7H SYM  SCANEOL
0475H SYM  RETRY
057CH SYM  SIMPLECOM
072BH SYM  ENDCOM
```