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


   1          LOAD:
              DO;
                  /* C P / M   C O M M A N D   F I L E   L O A D E R

                  COPYRIGHT (C) 1976, 1977, 1978
                  DIGITAL RESEARCH
                  BOX 579 PACIFIC GROVE
                  CALIFORNIA 93950

                  */

   2   1      DECLARE
                  TPA   LITERALLY '0100H',   /* TRANSIENT PROGRAM AREA */
                  DFCBA LITERALLY '005CH',   /* DEFAULT FILE CONTROL BLOCK */
                  DBUFF LITERALLY '0080H';   /* DEFAULT BUFFER ADDRESS */

              /* JMP LOADCOM TO START LOAD */
   3   1      DECLARE JUMP BYTE DATA(0C3H);
   4   1      DECLARE JUMPA ADDRESS DATA(.LOADCOM);

   5   1      DECLARE COPYRIGHT(*) BYTE DATA
                  (' COPYRIGHT (C) 1978, DIGITAL RESEARCH ');

   6   1      MON1: PROCEDURE(F,A) EXTERNAL;
   7   2          DECLARE F BYTE, A ADDRESS;
   8   2          END MON1;

   9   1      MON2: PROCEDURE(F,A) BYTE EXTERNAL;
  10   2          DECLARE F BYTE, A ADDRESS;
  11   2          END MON2;

  12   1      DECLARE SP ADDRESS;

  13   1      BOOT: PROCEDURE;
  14   2          STACKPTR = SP;
  15   2          RETURN;
  16   2          END BOOT;


  17   1      LOADCOM: PROCEDURE;
  18   2          DECLARE FCB (33) BYTE AT (DFCBA),
                     FCBA LITERALLY 'DFCBA';
  19   2          DECLARE BUFFER (128) BYTE AT (DBUFF),
                      BUFFA LITERALLY 'DBUFF';
  20   2          DECLARE SFCB(33) BYTE,  /* SOURCE FILE CONTROL BLOCK */
                      BSIZE LITERALLY '1024',
                      EOFILE LITERALLY '1AH',
                      SBUFF(BSIZE) BYTE,  /* SOURCE FILE BUFFER */
                      RFLAG BYTE,         /* READER FLAG */
                      SBP ADDRESS;        /* SOURCE FILE BUFFER POINTER */
```

```
                    /* LOADCOM LOADS TRANSIENT COMMAND FILES TO THE DISK FROM THE
                       CURRENTLY DEFINED READER PERIPHERAL.  THE LOADER PLACES THE MA
                -   CHINE
                       CODE INTO A FILE WHICH APPEARS IN THE LOADCOM COMMAND */

 21   2         DECLARE
                    TRUE LITERALLY '1',
                    FALSE LITERALLY '0',
                    FOREVER LITERALLY 'WHILE TRUE',
                    CR LITERALLY '13',
                    LF LITERALLY '10',
                    WHAT LITERALLY '63';

 22   2         PRINTCHAR: PROCEDURE(CHAR);
 23   3             DECLARE CHAR BYTE;
 24   3             CALL MON1(2,CHAR);
 25   3             END PRINTCHAR;

 26   2         CRLF: PROCEDURE;
 27   3             CALL PRINTCHAR(CR);
 28   3             CALL PRINTCHAR(LF);
 29   3             END CRLF;

 30   2         PRINTNIB: PROCEDURE(N);
 31   3             DECLARE N BYTE;
 32   3                 IF N > 9 THEN CALL PRINTCHAR(N+'A'-10); ELSE
 34   3                 CALL PRINTCHAR(N+'0');
 35   3             END PRINTNIB;

 36   2         PRINTHEX: PROCEDURE(B);
 37   3             DECLARE B BYTE;
 38   3                 CALL PRINTNIB(SHR(B,4)); CALL PRINTNIB(B AND 0FH);
 40   3             END PRINTHEX;

 41   2         PRINTADDR: PROCEDURE(A);
 42   3             DECLARE A ADDRESS;
 43   3             CALL PRINTHEX(HIGH(A)); CALL PRINTHEX(LOW(A));
 45   3             END PRINTADDR;

 46   2         PRINTM: PROCEDURE(A);
 47   3             DECLARE A ADDRESS;
 48   3             CALL MON1(9,A);
 49   3             END PRINTM;

 50   2         PRINT: PROCEDURE(A);
 51   3             DECLARE A ADDRESS;
                    /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE
                    NEXT DOLLAR SIGN IS ENCOUNTERED WITH PRECEDING CRLF */
 52   3             CALL CRLF;
 53   3             CALL PRINTM(A);
 54   3             END PRINT;

 55   2         DECLARE LA ADDRESS;  /* CURRENT LOAD ADDRESS */

 56   2         PERROR: PROCEDURE(A);
                    /* PRINT ERROR MESSAGE */
 57   3             DECLARE A ADDRESS;
```

```
 58   3          CALL PRINT(.('ERROR: $'));
 59   3          CALL PRINTM(A);
 60   3          CALL PRINTM(.(', LOAD ADDRESS $'));
 61   3          CALL PRINTADDR(LA);
 62   3          CALL BOOT;
 63   3          END PERROR;

 64   2      DECLARE DCNT BYTE;

 65   2      OPEN: PROCEDURE(FCB);
 66   3          DECLARE FCB ADDRESS;
 67   3          DCNT = MON2(15,FCB);
 68   3          END OPEN;

 69   2      CLOSE: PROCEDURE(FCB);
 70   3          DECLARE FCB ADDRESS;
 71   3          DCNT = MON2(16,FCB);
 72   3          END CLOSE;

 73   2      SEARCH: PROCEDURE(FCB);
 74   3          DECLARE FCB ADDRESS;
 75   3          DCNT = MON2(17,FCB);
 76   3          END SEARCH;

 77   2      SEARCHN: PROCEDURE;
 78   3          DCNT = MON2(18,0);
 79   3          END SEARCHN;

 80   2      DELETE: PROCEDURE(FCB);
 81   3          DECLARE FCB ADDRESS;
 82   3          CALL MON1(19,FCB);
 83   3          END DELETE;

 84   2      DISKREAD: PROCEDURE(FCB) BYTE;
 85   3          DECLARE FCB ADDRESS;
 86   3          RETURN MON2(20,FCB);
 87   3          END DISKREAD;

 88   2      DISKWRITE: PROCEDURE(FCB) BYTE;
 89   3          DECLARE FCB ADDRESS;
 90   3          RETURN MON2(21,FCB);
 91   3          END DISKWRITE;

 92   2      MAKE: PROCEDURE(FCB);
 93   3          DECLARE FCB ADDRESS;
 94   3          DCNT = MON2(22,FCB);
 95   3          END MAKE;

 96   2      RENAME: PROCEDURE(FCB);
 97   3          DECLARE FCB ADDRESS;
 98   3          CALL MON1(23,FCB);
 99   3          END RENAME;

100   2          MOVE: PROCEDURE(S,D,N);
101   3              DECLARE (S,D) ADDRESS, N BYTE,
                      A BASED S BYTE, B BASED D BYTE;
102   3                  DO WHILE (N:=N-1) <> 255;
```

```
103   4                 B = A; S=S+1; D=D+1;
106   4                 END;
107   3             END MOVE;

108   2         GETCHAR: PROCEDURE BYTE;
                    /* GET NEXT CHARACTER */
109   3             DECLARE I BYTE;
110   3             IF (SBP := SBP+1) <= LAST(SBUFF) THEN
111   3                 R TURN SBUFF(SBP);
                    /* OTHERWISE READ ANOTHER BUFFER FULL */
112   3                 DO SBP = 0 TO LAST(SBUFF) BY 128;
113   4                 IF (I:=DISKREAD(.SFCB)) = 0 THEN
114   4                     CALL MOVE(80H,.SBUFF(SBP),80H); ELSE
115   4                     DO;
116   5                     IF I<>1 THEN CALL PERROR(.('DISK READ$'));
118   5                     SBUFF(SBP) = EOFILE;
119   5                     SBP = LAST(SBUFF);
120   5                     END;
121   4                 END;
122   3             SBP = 0; RETURN SBUFF(0);
124   3             END GETCHAR;
125   2     DECLARE
            STACKPOINTER LITERALLY 'STACKPTR';

           /* INTEL HEX FORMAT LOADER */

126   2     RELOC: PROCEDURE;
127   3         DECLARE (RL, CS, RT) BYTE;
128   3         DECLARE
                TA ADDRESS,     /* TEMP ADDRESS */
                SA ADDRESS,     /* START ADDRESS */
                FA ADDRESS,     /* FINAL ADDRESS */
                NB ADDRESS,     /* NUMBER OF BYTES LOADED */

              MBUFF(256) BYTE,
              P BYTE,
              L ADDRESS;

129   3         SETMEM: PROCEDURE(B);
                    /* SET MBUFF TO B AT LOCATION LA MOD LENGTH(MBUFF) */
130   4             DECLARE (B,I) BYTE;
131   4             IF LA < L THEN
132   4                 CALL PERROR(.('INVERTED LOAD ADDRESS$'));
133   4                 DO WHILE LA > L + LAST(MBUFF); /* WRITE A PARAGRAPH */
134   5                     DO I = 0 TO 127; /* COPY INTO BUFFER */
135   6                     BUFFER(I) = MBUFF(LOW(L)); L = L + 1;
137   6                     END;
                        /* WRITE BUFFER ONTO DISK */
138   5                 P = P + 1;
139   5                 IF DISKWRITE(FCBA) <> 0 THEN
140   5                     DO; CALL PERROR(.('DISK WRITE$'));
142   6                     END;
143   5                 END;
144   4             MBUFF(LOW(LA)) = B;
145   4             END SETMEM;

146   3         DIAGNOSE: PROCEDURE;
```

```
147   4          DECLARE M BASED TA BYTE;

148   4          NEWLINE: PROCEDURE;
149   5              CALL CRLF; CALL PRINTADDR(TA); CALL PRINTCHAR(':');
152   5              CALL PRINTCHAR(' ');
153   5              END NEWLINE;

                 /* PRINT DIAGNOSTIC INFO MATION AT THE CONSOLE */
154   4              CALL PRINT(.('LOAD  ADDRESS $')); CALL PRINTADDR(TA);
156   4              CALL PRINT(.('ERROR ADDRESS $')); CALL PRINTADDR(LA);

158   4              CALL PRINT(.('BYTES READ:$')); CALL NEWLINE;
160   4              DO WHILE TA < LA;
161   5              IF (LOW(TA) AND 0FH) = 0  THEN CALL NEWLINE;
163   5              CALL PRINTHEX(MBUFF(TA-L)); TA=TA+1;
165   5              CALL PRINTCHAR(' ');
166   5              END;
167   4          CALL CRLF;
168   4          CALL BOOT;
169   4          END DIAGNOSE;

170   3          READHEX: PROCEDURE BYTE;
                     /* READ ONE HEX CHARACTER FROM THE INPUT */
171   4              DECLARE H BYTE;
172   4              IF (H := GETCHAR) - '0' <= 9 THEN RETURN H - '0';
174   4              IF H - 'A' > 5 THEN
175   4                  DO; CALL PRINT(.('INVALID HEX DIGIT$'));
177   5                  CALL DIAGNOSE;
178   5                  END;
179   4              RETURN H - 'A' + 10;
180   4              END READHEX;

181   3          READBYTE: PROCEDURE BYTE;
                     /* READ TWO HEX DIGITS */
182   4              RETURN SHL(READHEX,4) OR READHEX;
183   4              END READBYTE;

184   3          READCS: PROCEDURE BYTE;
                     /* READ BYTE WHILE COMPUTING CHECKSUM */
185   4              DECLARE B BYTE;
186   4              CS = CS + (B := READBYTE);
187   4              RETURN B;
188   4              END READCS;

189   3          MAKE$DOUBLE: PROCEDURE(H,L) ADDRESS;
                     /* CREATE A BOUBLE BYTE VALUE FROM TWO SINGLE BYTES */
190   4              DECLARE (H,L) BYTE;
191   4              RETURN SHL(DOUBLE(H),8) OR L;
192   4              END MAKE$DOUBLE;


                 /* INITIALIZE */
193   3          SA, FA, NB = 0;
194   3          P = 0; /* PARAGRAPH COUNT */
195   3          TA,L = TPA;  /* BASE ADDRESS OF TRANSIENT ROUTINES */
196   3          SBUFF(0) = EOFILE;
```

```
                /* READ RECORDS UNTIL :00XXXX IS ENCOUNTERED */

197   3             DO FOREVER;
                    /* SCAN THE : */
198   4               DO WHILE GETCHAR <> ':';
199   5               END;

                    /* SET CHECK SUM TO ZERO, AND SAVE THE RECORD LENGTH */
200   4             CS = 0;
                    /* MAY BE THE END OF TAPE */
201   4             IF (RL := READCS) = 0 THEN
202   4                   GO TO FIN;
203   4             NB = NB + RL;

204   4             TA, LA = MAKE$DOUBLE(READCS,READCS);
205   4             IF SA = 0 THEN SA = LA;


                    /* READ THE RECORD TYPE (NOT CURRENTLY USED) */
207   4             RT = READCS;

                    /* PROCESS EACH BYTE */
208   4               DO WHILE (RL := RL - 1) <> 255;
209   5               CALL SETMEM(READCS); LA = LA+1;
211   5               END;
212   4             IF LA > FA THEN FA = LA - 1;

                    /* NOW READ CHECKSUM AND COMPARE */
214   4             IF CS + READBYTE <> 0 THEN
215   4               DO; CALL PRINT(.('CHECK SUM ERROR $'));
217   5               CALL DIAGNOSE;
218   5               END;
219   4             END;

220   3           FIN:
                  /* EMPTY THE BUFFERS */
                  TA = LA;
221   3               DO WHILE L < TA;
222   4               CALL SETMEM(0); LA = LA+1;
224   4               END;
                  /* PRINT FINAL STATISTICS */
225   3           CALL PRINT(.('FIRST ADDRESS $')); CALL PRINTADDR(SA);
227   3           CALL PRINT(.('LAST  ADDRESS $')); CALL PRINTADDR(FA);
229   3           CALL PRINT(.('BYTES READ    $')); CALL PRINTADDR(NB);
231   3           CALL PRINT(.('RECORDS WRITTEN $')); CALL PRINTHEX(P);
233   3           CALL CRLF;

234   3           END RELOC;

              /* ARRIVE HERE FROM THE SYSTEM MONITOR, READY TO READ THE HEX TAP
          -   E */

               /* SET UP STACKPOINTER IN THE LOCAL AREA */
235   2          DECLARE STACK(16) ADDRESS;
236   2          SP = STACKPOINTER; STACKPOINTER = .STACK(LENGTH(STACK));
```

```
 238   2      LA = TPA;

 239   2      SBP = LENGTH(SBUFF);
                 /* SET UP THE SOURCE FILE */
 240   2        CALL MOVE(FCBA,.SFCB,33);
 241   2        CALL MOVE(.('HEX',0),.SFCB(9),4);
 242   2        CALL OPEN(.SFCB);
 243   2        IF DCNT = 255 THEN CALL PERROR(.('CANNOT OPEN SOURCE$'));

 245   2        CALL MOVE(.('COM'),FCBA+9,3);

              /* REMOVE ANY EXISTING FILE BY THIS NAME */
 246   2      CALL DELETE(FCBA);
              /* THEN OPEN A NEW FILE */
 247   2      CALL MAKE(FCBA); CALL OPEN(FCBA);
 249   2      IF DCNT = 255 THEN CALL PERROR(.('NO MORE DIRECTORY SPACE$')); EL
         -    SE
 251   2             DO; CALL RELOC;
 253   3             CALL CLOSE(FCBA);
 254   3             IF DCNT = 255 THEN CALL PERROR(.('CANNOT CLOSE FILE$'));
 256   3             END;
 257   2      CALL CRLF;

 258   2      CALL BOOT;
 259   2      END LOADCOM;
 260   1      END;
              EOF
```


MODULE INFORMATION:

```
     CODE AREA SIZE     = 0663H   1635D
     VARIABLE AREA SIZE = 057EH   1406D
     MAXIMUM STACK SIZE = 001AH     26D
     360 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
0000 LOAD#
0000 LOAD#
023B 13     023B 14     023F 15     0240 16     0240 17
02D0 22     02D4 24     02DF 25     02E0 26     02E0 27
02E5 28     02EA 29     02EB 30     02EF 32     02F8 33
0306 34     030F 35     0310 36     0314 38     0321 39
032A 40     032B 41     0331 43     0339 44     0341 45
0342 46     0348 48     0351 49     0352 50     0358 52
035B 53     0363 54     0364 56     036A 58     0370 59
0378 60     037E 61     0386 62     0389 63     038A 65
0390 67     039C 68     039D 69     03A3 71     03AF 72
03B0 73     03B6 75     03C2 76     03C3 77     03C3 78
03CE 79     03CF 80     03D5 82     03DE 83     03DF 84
03E5 86     03EF 87     03EF 88     03F5 90     03FF 91
03FF 92     0405 94     0411 95     0412 96     0418 98
0421 99     0422 100    0431 102    043D 103    0447 104
044E 105    0455 106    0458 107    0459 108    0459 110
0469 111    0472 112    0497 113    04A5 114    04BA 116
04C2 117    04C8 118    04D1 119    04D7 120    04D7 121
04DA 122    04E0 123    04E4 124    04E4 126    05FD 129
0601 131    060D 132    0613 133    0624 134    0632 135
064A 136    0651 137    0658 138    065C 139    0667 141
066D 142    066D 143    0670 144    067F 145    0680 146
06E6 148    06E6 149    06E9 150    06F1 151    06F6 152
06FB 153    0680 154    0686 155    068E 156    0694 157
069C 158    06A2 159    06A5 160    06B1 161    06BC 162
06BF 163    06D0 164    06D7 165    06DC 166    06DF 167
06E2 168    06E5 169    06FC 170    06FC 172    070B 173
0711 174    071D 176    0723 177    0726 178    0726 179
072E 180    072E 181    072E 182    073D 183    073D 184
073D 186    0748 187    074C 188    074C 189    0752 191
0763 192    04E4 193    04F0 194    04F4 195    04FD 196
0502 197    0502 198    050A 199    050D 200    0512 201
051D 202    0520 203    052E 204    0541 205    054D 206
0553 207    0559 208    0565 209    056C 210    0573 211
0576 212    0582 213    0589 214    0595 216    059B 217
059E 218    059E 219    05A1 220    05A7 221    05B3 222
05B8 223    05BF 224    05C2 225    05C8 226    05D0 227
05D6 228    05DE 229    05E4 230    05EC 231    05F2 232
05F9 233    05FC 234    0240 236    0247 237    024B 238
0251 239    0257 240    0263 241    026F 242    0275 243
027D 244    0283 245    028F 246    0295 247    029B 248
02A1 249    02A9 250    02B2 252    02B5 253    02BB 254
02C3 255    02C9 256    02C9 257    02CC 258    02CF 259
0000 MODULE#
```