

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

```

1      sub:
2      do;
3      /* modified 7/26/79 to work with cpm 2.0, module number not zero */
4      /
5      declare
6          wboot literally '0000h', /* warm start entry point */
7          bdos literally '0005h', /* jmp bdos */
8          dfcba literally '005ch', /* default fcb address */
9          dbuff literally '0080h'; /* default buffer address */

10     declare jump byte data(0c3h); /* c3 = jmp */
11     declare jadr address data(.submit);
12     /* jmp to submit is placed at the beginning of the module */

13     boot: procedure external;
14         /* system reboot */
15     end boot;

16     mon1: procedure(f,a) external;
17     declare f byte, a address;
18     /* bdos interface, no returned value */
19     end mon1;

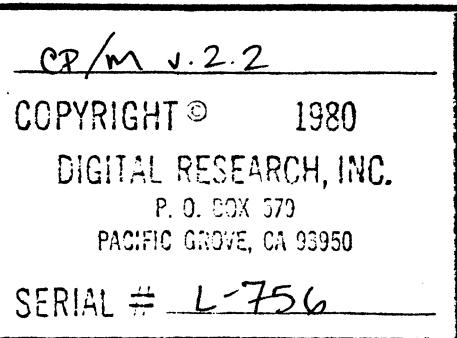
20     mon2: procedure(f,a) byte external;
21     declare f byte, a address;
22     /* bdos interface, return byte value */
23     end mon2;

24     declare
25         copyright(*) byte data
26             ('copyright(c) 1977, digital research');

27     declare
28         ln(5) byte initial('001 $'),
29         ln1 byte at(.ln(0)),
30         ln2 byte at(.ln(1)),
31         ln3 byte at(.ln(2)),
32         dfcb(33) byte initial(0,'$$$      SUB',0,0,0),
33         drec byte at(.dfcb(32)), /* current record */
34         buff(128) byte at(dbuff), /* default buffer */
35         sfcba(33) byte at(dfcba); /* default fcb */

36     submit: procedure;
37
38         /* the cp/m 'submit' function
39
40             copyright (c) 1976, 1977, 1978
41             digital research
42             box 579
43             pacific grove, ca.

```



```
93950
*/
16 2 declare lit literally 'literally',
      dcl lit 'declare',
      proc lit 'procedure',
      addr lit 'address',
      ctll lit 'Och',
      lca lit '110$0001b'; /* lower case a */
      lcZ lit '111$1010b'; /* lower case z */
      endfile lit '1ah'; /* cp/m end of file */

17 2 declare
      true literally '1',
      false literally '0',
      forever literally 'while true',
      cr literally '13',
      lf literally '10',
      what literally '63';

18 2 print: procedure(a);
19 3     declare a address;
      /* print the string starting at address a until the
       next dollar sign is encountered */
20 3     call mon1(9,a);
21 3     end print;

22 2 declare dcnt byte;

23 2 open: procedure(fcb);
24 3     declare fcb address;
25 3     dcnt = mon2(15,fcb);
26 3     end open;

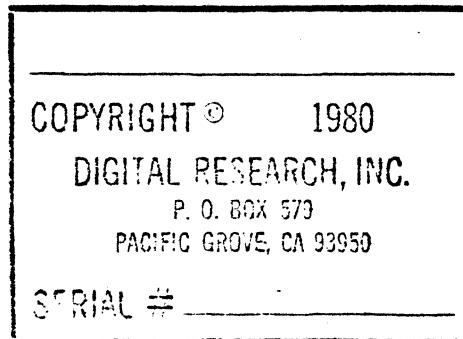
27 2 close: procedure(fcb);
28 3     declare fcb address;
29 3     dcnt = mon2(16,fcb);
30 3     end close;

31 2 delete: procedure(fcb);
32 3     declare fcb address;
33 3     call mon1(19,fcb);
34 3     end delete;

35 2 diskread: procedure(fcb) byte;
36 3     declare fcb address;
37 3     return mon2(20,fcb);
38 3     end diskread;

39 2 diskwrite: procedure(fcb) byte;
40 3     declare fcb address;
41 3     return mon2(21,fcb);
42 3     end diskwrite;

43 2 make: procedure(fcb);
44 3     declare fcb address;
45 3     dcnt = mon2(22,fcb);
```



```

46      end make;

47      move: procedure(s,d,n);
48          declare (s,d) address, n byte;
49          declare a based s byte, b based d byte;
50          do while (n := n - 1) <> 255;
51          b = a; s = s + 1; d = d + 1;
52          end;
53      end move;

54      declare oldsp address; /* calling program's stack pointer */

55      error: procedure(a);
56          declare a address;
57          call print(..(cr,lf,'$'));
58          call print(..('Error On Line $'));
59          call print(.ln1);
60          call print(a);
61          stackptr = oldsp;
62          /* return to ccp */
63          end error;

64      declare sstring(128) byte, /* substitute string */
65          sbp byte;             /* source buffer pointer (0-128) */

66      setup: procedure;
67          /* move buffer to substitute string */
68          call move(.buff(1),.sstring(0),127);
69          sstring(buff(0))=0; /* mark end of string */
70          call move(..('SUB'),.sfcb(9),3); /* set file type to sub */
71          call open(.sfcb(0));
72          if dcnt = 255 then
73              call error(..('No ``SUB'' File Present$'));
74          /* otherwise file is open - read subsequent data */
75          sbp = 128; /* causes read below */

76      end setup;

77      getsource: procedure byte;
78          /* read the next source character */
79          declare b byte;
80          if sbp > 127 then
81              do; if diskread(.sfcb(0)) <> 0 then
82                  return endfile;
83              sbp = 0;
84              end;
85              if (b := buff((sbp:=sbp+1)-1)) = cr then
86                  do; /* increment line */
87                  if (ln3 := ln3 + 1) > '9' then
88                      do; ln3 = '0';
89                      if (ln2 := ln2 + 1) > '9' then
90                          do; ln2 = '0';
91                          ln1 = ln1 + 1;
92                          end;
93                      end;

```

COPYRIGHT © 1980
 DIGITAL RESEARCH, INC.
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SERIAL # _____

```

94   4           end;
      /* translate to upper case */
95   3   if (b-61h) < 26 then /* lower case alpha */
96   3       b = b and 5fh; /* change to upper case */
97   3   return b;
98   3   end getsource;

99   2   writebuff: procedure;
      /* write the contents of the buffer to disk */
100  3   if diskwrite(.dfcb) <> 0 then /* error */
101  3       call error.(('Disk Write Error$'));
102  3   end writebuff;

103  2   declare rbuff(2048) byte, /* jcl buffer */
      rbp address,          /* jcl buffer pointer */
      rlen byte;           /* length of current command */

104  2   fillrbuff: procedure;
105  3   declare (s,ssbp) byte; /* sub string buffer pointer */

106  3   notend: procedure byte;
      /* look at next character in sstring, return
         true if not at the end of the string - char passed
         back in 's' */
107  4   if not ((s := sstring(ssbp)) = ' ' or s = 0) then
108  4       do;
109  5       sssbp = sssbp + 1;
110  5       return true;
111  5       end;
112  4       return false;
113  4   end notend;

114  3   deblankparm: procedure;
      /* clear to next non blank substitute string */
115  4   do while sstring(ssbp) = ' ';
116  5   sssbp = sssbp + 1;
117  5   end;
118  4   end deblankparm;

119  3   putrbuff: procedure(b);
120  4   declare b byte;
121  4   if (rbp := rbp + 1) > last(rbuff) then
122  4       call error.(('Command Buffer Overflow$'));
123  4   rbuff(rbp) = b;
      /* len: c1 ... c125 :00:$ = 128 chars */
124  4   if (rlen := rlen + 1) > 125 then
125  4       call error.(('Command Too Long$'));
126  4   end putrbuff;

127  3   declare (reading,b) byte;
      /* fill the jcl buffer */
128  3   rbuff(0),rbp = 0;
129  3   reading = true;
130  3   do while reading;
131  4       rlen = 0; /* reset command length */
132  4       do while (b:=getsource) <> endfile and b <> cr;
133  5       if b <> lf then

```

COPYRIGHT © 1980
 DIGITAL RESEARCH, INC.
 P. O. BOX 579
 PACIFIC GROVE, CA 93950
 SERIAL # _____

```

134   5           do; if b = '$' then /* copy substitute string */
136   6           do; if (b:=getsource) = '$' then
138   7               /* $$ replaced by $ */
139   7               call putrbuf(b); else
140   7               if (b := b - '0') > 9 then
141   7                   call error(.('Parameter Error$'));
142   8               do; /* find string 'b' in sstring */
143   8               sskb = 0; call deblankparm; /* ready to sca
144   8           - n sstring */
145   9           do while b <> 0; b = b - 1;
146  10             /* clear next parameter */
147  11             do while notend;
148  12             end;
149  13             call deblankparm;
149  13             end;
149  13             /* ready to copy substitute string from pos
150  14           - ition sskb */
151  15             do while notend;
152  16               call putrbuf(s);
153  17               end;
154  18               end;
155  19               if b = '^' then /* control character */
156  20                 do; /* must be ^a ... ^z */
157  21                 if (b:=getsource - 'a') > 25 then
158  22                     call error(.('Invalid Control Character$'))
159  23           - );
160  24             else
161  25               call putrbuf(b+1);
162  26             end; else /* not $ or ^ */
163  27               call putrbuf(b);
164  28             end;
165  29             end; /* of line or input file - compute length */
166  30             reading = b = cr;
167  31             call putrbuf(rlen); /* store length */
168  32             end;
169  33             /* entire file has been read and processed */
170  34           end fillrbuff;

171  2     makefile: procedure;
172  3         /* write resulting command file */
173  3         declare i byte;
174  3         getrbuff: procedure byte;
175  4             return rbuff(rbp := rbp - 1);
176  4             end getrbuff;

177  3         call delete(.dfcb);
178  3         drec = 0; /* zero the next record to write */
179  3         call make(.dfcb);
180  3         if dcnt = 255 then call error(.('Directory Full$'));
181  4             do while (i := getrbuff) <> 0;
182  4             /* copy i characters to buffer */
183  5             /* 00 $ at end of line gives 1.3 & 1.4 compatibility */
184  5             buff(0) = i; buff(i+1) = 00; buff(i+2) = '$';
185  5             do while i > 0;
186  5             buff(i) = getrbuff; i=i-1;
187  5             end;

```

COPYRIGHT © 1980
 DIGITAL RESEARCH, INC.
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SERIAL # _____

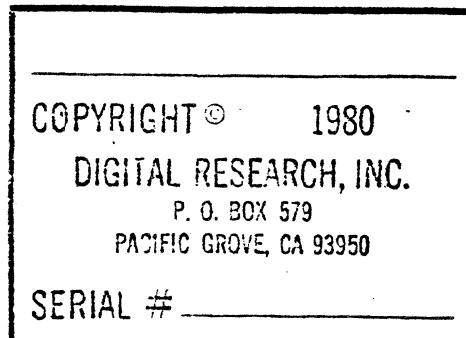
```
186   4           /* buffer filled to $ */
187   4           call writebuff;
188   3           end;
189   3           call close(.dfcb);
190   3           if dcnt = 255 then call error.(('Cannot Close, Read/Only?'));
191   3           end makefile;

192   2           /* enter here from the ccp with the fcb set */
193   2           declare stack(10) address; /* working stack */
194   2           oldsp = stackptr;
195   2           stackptr = .stack(length(stack));

196   2           call setup;
197   2           call fillrbuf;
198   2           call makefile;
199   2           call boot; /* reboot causes commands to be executed */
200   1           end submit;
200   1           end;
EOF
```

MODULE INFORMATION:

CODE AREA SIZE = 0487H 1159D
VARIABLE AREA SIZE = 08DDH 2269D
MAXIMUM STACK SIZE = 000AH 10D
294 LINES READ
0 PROGRAM ERROR(S)



0000 SUBMIT#

0000 SUB#

01DF 15	01F7 18	01FD 20	0206 21	0207 23
020D 25	0219 26	021A 27	0220 29	022C 30
022D 31	0233 33	023C 34	023D 35	0243 37
024D 38	024D 39	0253 41	025D 42	025D 43
0263 45	026F 46	0270 47	027F 50	028B 51
0295 52	029C 53	02A3 54	02A6 55	02A7 57
02AD 59	02B3 60	02B9 61	02BF 62	02C7 63
02CB 64	02CC 66	02CC 67	02D8 68	02E3 69
02EF 70	02F5 71	02FD 72	0303 73	0308 74
0309 75	0309 77	0312 79	031D 80	0320 81
0325 82	0325 83	033D 85	034B 87	0350 88
035B 90	0360 91	0362 92	0362 93	0362 94
0362 95	036C 96	0374 97	0378 98	0378 99
0378 100	0383 101	0389 102	038A 104	0481 106
0481 107	04A3 109	04A7 110	04AA 111	04AA 112
04AD 113	04D 114	04AD 115	04BC 116	04CO 117
04C3 118	04C4 119	04C8 121	04D8 122	04DE 123
04E9 124	04F7 125	04FD 126	038A 128	0395 129
039A 130	03A1 131	03A6 132	03C1 133	03C9 135
03D1 137	03DC 138	03E6 139	03F5 140	03FE 142
0403 143	0406 144	040E 145	0412 146	0419 147
041C 148	041F 149	0422 150	0429 151	0430 152
0433 153	0433 154	0436 155	043E 157	044D 158
0456 159	045E 160	0461 161	0468 162	0468 163
046B 164	0476 165	047D 166	0480 167	04FE 168
057A 170	057A 171	0587 172	04FE 173	0504 174
0509 175	050F 176	0517 177	051D 178	0528 179
052E 180	0537 181	0542 182	054B 183	0558 184
055C 185	055F 186	0562 187	0565 188	056B 189
0573 190	0579 191	01DF 193	01E6 194	01EA 195
01ED 196	01F0 197	01F3 198	01F6 199	

COPYRIGHT © 1980

DIGITAL RESEARCH, INC.

P. O. BOX 579
PACIFIC GROVE, CA 93950

SERIAL # _____

