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

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

  3    1     declare jump byte data(0c3h); /* c3 = jmp */
  4    1     declare jadr address data(.submit);
             /* jmp to submit is placed at the beginning of the module */

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

  7    1     mon1: procedure(f,a) external;
  8    2         declare f byte, a address;
                 /* bdos interface, no returned value */
  9    2         end mon1;

 10    1     mon2: procedure(f,a) byte external;
 11    2         declare f byte, a address;
                 /* bdos interface, return byte value */
 12    2         end mon2;

 13    1     declare
                 copyright(*) byte data
                     ( ' copyright(c) 1977, digital research ');

 14    1     declare
                 ln(5) byte initial('001 $'),
                 ln1 byte at(.ln(0)),
                 ln2 byte at(.ln(1)),
                 ln3 byte at(.ln(2)),
                 dfcb(33) byte initial(0,'$$$     SUB',0,0,0),
                 drec byte at(.dfcb(32)),  /* current record */
                 buff(128) byte at(dbuff),   /* default buffer */
                 sfcb(33)  byte at(dfcba);   /* default fcb */

 15    1     submit: procedure;

                 /* t h e    c p / m   's u b m i t'   f u n c t i o n

                     copyright (c) 1976, 1977, 1978
                     digital research
                     box 579
                     pacific grove, ca.
```

```
                    93950
                */
16   2      declare lit literally 'literally',
                dcl lit 'declare',
                proc lit 'procedure',
                addr lit 'address',
                ctll lit '0ch',
                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   3          end make;

47   2      move: procedure(s,d,n);
48   3          declare (s,d) address, n byte;
49   3          declare a based s byte, b based d byte;
50   3              do while (n := n - 1) <> 255;
51   4              b = a; s = s + 1; d = d + 1;
54   4              end;
55   3          end move;

56   2      declare oldsp address;   /* calling program's stack pointer */

57   2      error: procedure(a);
58   3          declare a address;
59   3          call print(.(cr,lf,'$'));
60   3          call print(.('Error On Line $'));
61   3          call print(.ln1);
62   3          call print(a);
63   3          stackptr = oldsp;
                 /* return to ccp */
64   3          end error;

65   2      declare sstring(128) byte, /* substitute string */
                 sbp byte;              /* source buffer pointer (0-128) */


66   2      setup: procedure;
                 /* move buffer to substitute string */
67   3          call move(.buff(1),.sstring(0),127);
68   3          sstring(buff(0))=0; /* mark end of string */
69   3          call move(.('SUB'),.sfcb(9),3); /* set file type to sub */
70   3          call open(.sfcb(0));
71   3          if dcnt = 255 then
72   3              call error(.('No ''SUB'' File Present$'));
                 /* otherwise file is open - read subsequent data */
73   3          sbp = 128; /* causes read below */

74   3          end setup;


75   2      getsource: procedure byte;
                 /* read the next source character */
76   3          declare b byte;
77   3          if sbp > 127 then
78   3              do; if diskread(.sfcb(0)) <> 0 then
80   4                  return endfile;
81   4              sbp = 0;
82   4              end;
83   3          if (b := buff((sbp:=sbp+1)-1)) = cr then
84   3              do; /* increment line */
85   4              if (ln3 := ln3 + 1) > '9' then
86   4                  do; ln3 = '0';
88   5                  if (ln2 := ln2 + 1) > '9' then
89   5                      do; ln2 = '0';
91   6                      ln1 = ln1 + 1;
92   6                      end;
93   5                  end;
```

```
 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) <>  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                      ssbp = ssbp + 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                      ssbp = ssbp + 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
```

```
134   5                    do; if b = '$' then /* copy substitute string */
136   6                       do; if (b:=getsource) = '$' then
                                  /* $$ replaced by $ */
138   7                             call putrbuff(b); else
139   7                       if (b := b - '0') > 9 then
140   7                             call error(.('Parameter Error$')); else
141   7                             do; /* find string 'b' in sstring */
142   8                             ssbp = 0; call deblankparm; /* ready to sca
          -   n sstring */
144   8                                  do while b <> 0; b = b - 1;
                                         /* clear next parameter */
146   9                                      do while notend;
147   10                                     end;
148   9                                  call deblankparm;
149   9                                  end;
                                  /* ready to copy substitute string from pos
          -   ition ssbp */
150   8                                  do while notend;
151   9                                  call putrbuff(s);
152   9                                  end;
153   8                             end;
154   7                       end; else /* not a '$' */
155   6                    if b = '^' then /* control character */
156   6                        do; /* must be ^a ... ^z */
157   7                        if (b:=getsource - 'a') > 25 then
158   7                            call error(.('Invalid Control Character$')
          -  );
                                 else
159   7                                call putrbuff(b+1);
160   7                        end; else /* not $ or ^ */
161   6                    call putrbuff(b);
162   6                    end;
163   5                end; /* of line or input file - compute length */
164   4             reading = b = cr;
165   4             call putrbuff(rlen); /* store length */
166   4             end;
                   /* entire file has been read and processed */
167   3          end fillrbuff;

168   2      makefile: procedure;
                   /* write resulting command file */
169   3          declare i byte;
170   3          getrbuff: procedure byte;
171   4              return rbuff(rbp := rbp - 1);
172   4              end getrbuff;

173   3          call delete(.dfcb);
174   3          drec = 0; /* zero the next record to write */
175   3          call make(.dfcb);
176   3          if dcnt = 255 then call error(.('Directory Full$'));
178   3              do while (i := getrbuff) <> 0;
                       /* copy i characters to buffer */
                       /* 00 $ at end of line gives 1.3 & 1.4 compatibility */
179   4                  buff(0) = i; buff(i+1) = 00; buff(i+2) = '$';
182   4                      do while i > 0;
183   5                      buff(i) = getrbuff; i=i-1;
185   5                      end;
```

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

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

 195   2            call setup;
 196   2            call fillrbuff;
 197   2            call makefile;
 198   2            call boot; /* reboot causes commands to be executed */
 199   2            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    04.D 114    04AD 115    04BC 116    04C0 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
```