DIGITAL RESEARCH

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

BASIC DISK OPERATING SYSTEM (BDOS)

CP/M VERSION _____

SER. # _____

```
 1>
 2>NDECLARE FDOS LITERALLY '3200H';
 3>     /*          CP/M BDOS
 4>                 COPYRIGHT (C) 1976
 5>                 DIGITAL RESEARCH
 6>                 BOX 579, PACIFIC GROVE
 7>                 CALIFORNIA, 93950
 8>     */
 9>FDOS: DECLARE BOOT LITERALLY 'UBOOT';
10>
11>
12>  /* C P / M   B A S I C   I / O   S Y S T E M   ( V 1 . 3 ) */
13>
14>NDECLARE BASE LITERALLY '3E00H';   /* DISK INTERFACE AND CONSOLE IO */
15>  /* THE FOLLOWING SUBROUTINES ARE ASSUMED TO EXIST,
16>     STARTING AT THE ADDRESS 'BASE'
17>
18>     BASE:     BOOT      SYSTEM REBOOT OPERATION
19>     BASE+3    WBOOT     SYSTEM REBOOT - WARM START
20>     BASE+6    CONSTAT   CONSOLE STATUS - RETURNS
21>                             0 IN REG-A IF NO CONSOLE DATA READY
22>                            FF IF CHARACTER IS READY
23>     BASE+9    CONIN     CONSOLE CHARACTER INTO ACCUMULATOR - 0 PARITY
24>     BASE+12   CONOUT    CONSOLE CHARACTER SENT FROM REGISTER C
25>     BASE+15   LIST      SEND CHARACTER FROM REGISTER C TO LIST DEVICE
26>     BASE+18   PUNCH     SEND CHARACTER FROM REGISTER C TO PUNCH DEVICE
27>     BASE+21   READER    READ CHARACTER TO REGISTER A WITH 0 PARITY
28>     BASE+24   HOME      MOVE DISK HEAD TO TRACK 0
29>     BASE+27   SELDSK    SELECT DISK DRIVE GIVEN BY REGISTER C (0,1,..)
30>     BASE+30   SETTRK    SET TRACK (0-76) GIVEN BY REGISTER C
31>     BASE+33   SETSEC    SET SECTOR NUMBER GIVEN BY REG C (1-26)
32>     BASE+36   SETDMA    SET DMA ADDRESS GIVEN BY REG PAIR B,C (INITIALL
33>Y
34>                         DEFAULTED TO 80H)
35>     BASE+39   READ      READ DISK SECTOR (SETTRK, SETSEC, SELDSK ASSUME
36>ND)
37>                             ERROR RETURNS INN REGISTER A IN THREE
38>                             LEAST SIGNIFICANT BITS (2,1, AND 0)
39>                             BIT       ERROR
40>                              0        HARDWARE MALFUNCTION
41>                              1        DRIVE NOT READY
42>                              2        COMMAND SEQUENCE ERROR
43>     BASE+42   WRITE     WRITE DISK SECTOR (SETTRK, SELDSK ASSUMED)
44>                             ERROR RETURNS IN REGISTER A AS ABOVE
45>
46>     CP/M ALSO PROVIDES A TEN BYTE AREA IMMEDIATELY AHEAD OF THE
47>     DISK AND CONSOLE INTERFACE FOR TEMPORARY STORAGE IN CASE THE
48>     INTERFACE IS IMPLEMENTED IN ROM
49>     */
50>
51>
52>
53>
54>NDISKMON: PROCEDURE(FUNC,INFO) ADDRESS;
55>     DECLARE COPYRIGHT DATA(
56>         'COPYRIGHT (C) 1976, DIGITAL RESEARCH');
57>
58>     DECLARE FUNC BYTE,
59>        LINFO BYTE,       /* LOW ORDER INFO */
60>        INFO ADDRESS,
```

CP/M VERSION 1.3 1.8 → 11E
COPYRIGHT © 1976
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA. 93950
SER. # BDOS

```
61>        ARET ADDRESS, RET BYTE,
62>
63>     /* FUNC IS THE DISK MONITOR FUNCTION NUMBER AS SHOWN BELOW:
64>          0:   SYSTEM RESET
65>          1:   READ CONSOLE DEVICE
66>          2:   WRITE CONSOLE DEVICE
67>          3:   READ READER DEVICE
68>          4:   WRITE PUNCH DEVICE
69>          5:   WRITE LIST DEVICE
70>          6:   INTERROGATE MEMORY SIZE
71>          7:   INTERROGATE DEVICE STATUS
72>          8:   CHANGE DEVICE STATUS
73>          9:   PRINT BUFFER ON CONSOLE
74>         10:   READ BUFFER FROM CONSOLE
75>         11:   CONSOLE CHARACTER READY
76>         12:   LIFT HEAD (NO OPERATION ON CPM 16D2JUN75)
77>         13:   RESET DISK SYSTEM - SELECT DISK 0
78>         14:   SELECT DISK 'INFO'
79>         15:   OPENN FILE
80>         16:   CLOSE FILE
81>         17:   SEARCH FOR FIRST OCCURRENCE
82>         18:   SEARCH FOR NEXT OCCURRENCE
83>         19:   DELETE A FILE
84>         20:   READ A FILE
85>         21:   WRITE A FILE
86>         22:   CREATE A FILE
87>         23:   RENAME A FILE
88>         24:   RETURN LOGIN VECTOR - EACH BIT CORRESPONDS TO
89>               A DISK NUMBER, FROM LSB TO MSB.  1 INDICATES
90>               THE DISK IS LOGGED IN.
91>         25:   RETURN CURRENTLY SELECTED DISK NUMBER
92>         26:   SET SUBSEQUENT DMA ADDRESS
93>         27:   RETURNN BASE ADDRESS OF ALLOCATION VECTOR
94>               (USED TO DETERMINE REMAINING SPACE)
95>     */
96>
97>NDECLARE
98>     EQU LITERALLY 'LITERALLY',
99>     BOOTF     EQU  '3E#00H',
100>    WBOOT     EQU  '3E#03H',
101>    CONSF     EQU  '3E#06H',
102>    CONIF     EQU  '3E#09H',
103>    CONOF     EQU  '3E#0CH',
104>    LISTF     EQU  '3E#0FH',
105>    PUNF      EQU  '3E#12H',
106>    READF     EQU  '3E#15H',
107>    HOMF      EQU  '3E#18H',
108>    SELF      EQU  '3E#1BH',
109>    TRKF      EQU  '3E#1EH',
110>    SECF      EQU  '3E#21H',
111>    DMAF      EQU  '3E#24H',
112>    DRDF      EQU  '3E#27H',
113>    DWRF      EQU  '3E#2AH';
114>
115>NDECLARE TRUE LITERALLY '1',
116>     FALSE LITERALLY '0';
117>
118>NDECLARE CHAR#RDY BYTE INITIAL(FALSE), /* TRUE IF CHAR READ */
119>     KB#CHAR BYTE, /* VALUE OF CHARACTER WHEN CHAR#RDY IS TRUE */
120>     LISTCOPY BYTE; /* TRUE IF COPYING TO LIST DEVICE */
```

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

```
121)
122)CONRDY: PROCEDURE BYTE;
123)     /* RETURN TRUE IF CHAR READY AT CONSOLE */
124)     GO TO CONSF;
125)     END CONRDY;
126)
127)CONIN: PROCEDURE BYTE;
128)     /* READ NEXT CONSOLE CHARACTER */
129)     IF CHAR$RDY THEN /* CHARACTER IS READY */
130)         DO; CHAR$RDY = FALSE; RETURN KB$CHAR;
131)         END;
132)     /* OTHERWISE READ THE CHARACTER */
133)     GO TO CONIF;
134)     END CONIN;
135)
136)CONBRK: PROCEDURE BYTE;
137)     DECLARE CTLC LITERALLY '03H';
138)     DECLARE CTLS LITERALLY '13H';
139)     IF CHAR$RDY THEN RETURN TRUE; /* CHARACTER ALREADY READ */
140)     IF CONRDY THEN /* CHECK FOR TYPE TERMINATION FUNCTION */
141)         DO;
142)         IF (KB$CHAR := CONIN) = CTLS THEN /* STOP TYPE */
143)             DO; IF CONIN = CTLC THEN GO TO BOOT;
144)             RETURN FALSE;
145)             END;
146)         RETURN (CHAR$RDY := TRUE);
147)         END;
148)     RETURN FALSE;
149)     END CONBRK;
150)
151)CONCHAR: PROCEDURE(CHAR);
152)     DECLARE CHAR BYTE;
153)     GO TO CONCF;
154)     END CONCHAR;
155)
156)CONOUT: PROCEDURE(CHAR);
157)     DECLARE CHAR BYTE;
158)     /* CHECK FOR BREAK CHARACTER */
159)     IF CONBRK THEN ;
160)     /* SEND CONSOLE CHARACTER */
161)     CALL CONCHAR(CHAR);
162)     END CONOUT;
163)
164)LSTOUT: PROCEDURE(CHAR);
165)     DECLARE CHAR BYTE;
166)     GO TO LISTF;
167)     END LSTOUT;
168)
169)PUNOUT: PROCEDURE(CHAR);
170)     DECLARE CHAR BYTE;
171)     GO TO PUNF;
172)     END PUNOUT;
173)
174)READIN: PROCEDURE BYTE;
175)     GO TO READF;
176)     END READIN;
177)
178)TRACK0: PROCEDURE;
179)     GO TO HOMF;
180)     END TRACK0;
```



```
181)
182)SELDISK: PROCEDURE(DISK);
183)     DECLARE DISK BYTE;
184)     GO TO SELF;
185)     END SELDISK;
186)
187)SELTRK: PROCEDURE(TRACK);
188)     DECLARE TRACK BYTE;
189)     GO TO TRKF;
190)     END SELTRK;
191)
192)SELSEC: PROCEDURE(SECTOR);
193)     DECLARE SECTOR BYTE;
194)     GO TO SECF;
195)     BCBBSSNPORTHUNCBJUTDHORTPOHUSSSSH
CBSHHSSQCBSSSNCBSSSSBYTE;
196)     GO TO DRDF;
197)     END READ$DISK;
198)
199)WRITE$DISK: PROCEDURE BYTE;
200)     GO TO DWRF;
201)     END WRITE$DISK;
202)
203)  /* CONSOLE COMMUNICATIONN PROCEDURES */
204)
205)DDECLARE
206)     /* SPECIAL CHARACTERS */
207)     ALT EQU '7DH',
208)     ESC EQU '1BH',
209)     TAB EQU '09H',
210)     BEL EQU '07H',
211)     LF  EQU '10',
212)     CR  EQU '13';
213)
214)NDECLARE COLUMN BYTE INITIAL(0), /* CURRENT CONSOLE COLUMN */
215)
216)
217)NDECLARE IOSTATA ADDRESS INITIAL(3),  /* IO STATUS BYTE LOCATION */
218)     IOSTAT BASED IOSTATA BYTE,       /* VALUE OF STATUS BYTE */
219)     /* IOSTAT DEFINES THE CURRENT DEVICE ASSIGNMENT
220)     0-1 CONSOLE
221)         0 TTY
222)         1 CRT
223)         2 BATCH (USE READER DEFINITION)
224)         3 USER (1)
225)     2-3 READER
226)         0 TTY
227)         1 PTR
228)         2 USER (1)
229)         3 USER (2)
230)     4-5 PUNCH
231)         0 TTY
232)         1 PTP
233)         2 USER (1)
234)         3 USER (2)
235)     6-7 LIST
236)         0 TTY
237)         1 CRT
238)         2 USER (1)
239)         3 USER (2)
240)     */
```

```
241)
242)TABOUT: PROCEDURE(CHAR);
243)     DECLARE (I,CHAR) BYTE;
244)     I = (CHAR = TAB AND (7 - (COLUMN AND 7)));
245)     IF CHAR = TAB THEN CHAR = ' ';
246)         DO WHILE (I := I - 1) <>  254;
247)         IF CHAR = CR THEN COLUMN = 0;
248)         IF CHAR >= ' ' THEN COLUMN=COLUMN+1;
249)         CALL CONOUT(CHAR);
250)         IF LISTCOPY THEN CALL LSTOUT(CHAR);
251)         END;
252)     END TABOUT;
253)
254)
255)CRLF: PROCEDURE;
256)     CALL TABOUT(CR);
257)     CALL TABOUT(LF);
258)     END CRLF;
259)
260)PPRINT: PROCEDURE(A);
261)     DECLARE A ADDRESS, (I, M BASED A) BYTE;
262)     /* PRINT THE STRING STARTING AT ADDRESS A UNTIL THE NEXT
263)     OCCURRENCE OF A DOLLAR SIGN */
264)         DO WHILE (I:=M) <>  '$'; CALL TABOUT(I);
265)         A = A + 1;
266)         END;
267)     END PRINT;
268)
269)READ: PROCEDURE;
270)     /* READ CHARACTERS FROM THE CONSOLE DEVICE
271)     INTO THE MEMORY LOCATION GIVEN BY 'INFO',
272)     UNTIL THE FIRST CARRIAGE RETURN
273)     IS ENCOUNTERED.  ALLOW BACKSPACE (RUBOUT),
274)     LINE ELIMINATE (CTL U), AND SYSTEM RE-BOOT
275)     (CTL C) */
276)     DECLARE
277)         SLASH EQU '5CH',
278)         CTLC  EQU '03H',
279)         CTLU  EQU '15H',
280)         CTL   EQU '5EH',
281)         CTLE  EQU '05H',
282)         CTLP  EQU '10H',
283)         CTLZ  EQU '1AH',
284)         CTLL  EQU '0CH';
285)
286)     /*  THE INFO POINTER IS ASSUMED TO ADDRESS AN
287)     AREA OF MEMORY CONTAINING TWO BYTE QUANTITIES.
288)     THE FIRST GIVES THE MAXIMUM BUFFER LENGTH, AND
289)     THE SECOND IS SET TO THE NUMBER OF CHARACTERS
290)     SCANNED UPON RETURN */
291)
292)     DECLARE MAXL BASED INFO BYTE,  /* MAX LENGTH */
293)         COMLEN BYTE,               /* SCANNED LENGTH */
294)         BUFFER BASED INFO BYTE,    /* BUFFER */
295)         C BYTE;
296)
297)     CTLOUT: PROCEDURE;
298)         /* PRINT UP-ARROW IN FRONT OF LAST CHARACTER READ */
299)         CALL TABOUT(CTL); CALL TABOUT(C OR 40H);
300)         END CTLOUT;
```



```
301)
302)     COMLEN = 0;
303)         DO WHILE COMLENN < MAXL;
304)          /* MAKE ALPHABETICS UPPER CASE */
305)         IF (C := CONIN) = CTLC THEN
306)             DO; CALL CTLOUT; CALL CRLF;
307)             GO TO BOOT;
308)             END;  ELSE
309)         IF C = CTLE THEN /* PHYSICAL RETURN */
310)             CALL CRLF; ELSE
311)         IF C = CTLP THENN LISTCOPY = NOT LISTCOPY; ELSE
312)        *IF C = CR THEN
313)             DO; BUFFER(1) = COMLEN;
314)             CALL TABOUT(CR);
315)             RETURN;
316)             END;  ELSE
317)         IF C = CTLU THEN
318)             DO; CALL CTLOUT; CALL CRLF; COMLEN=0;
319)             END; ELSE
320)         IF C = 7FH THENN /* RUBOUT */
321)             DO;
322)             IF COMLEN >  0 THEN
323)                 CALL TABOUT(BUFFER((COMLEN:=COMLEN-1)+2));
324)             END; ELSE
325)             DO;
326)                 IF (C AND 01100000B) = 0 THEN /* CONTROL CHARACTER */
327)                     CALL CTLOUT; ELSE
328)                 IF C = TAB THEN CALL TABOUT(TAB); ELSE
329)                     CALL TABOUT(C);
330)                 BUFFER ((COMLEN:=COMLEN+1)+1) = C;
331)             END;
332)         END;
333)     END READ;
334)
335)
336) DECLARE MAXDSK EQU '1',   /* MAX DISK NUMBER 0,1,... */
337)     NDISK EQU '2';       /* NUMBER OF DISKS = MAXDSK+1 */
338)
339) DECLARE (DPTR,DCNT) BYTE,
340)     BUFFA ADDRESS INITIAL(80H),
341)     BUFF     BASED BUFFA (128) BYTE;
342)
343) DECLARE DMX EQU '63',
344)         /* DMX IS THE LAST DIRECTORY ENTRY NUMBER
345)         (LISTED AS 0, 1, ... , DMX) */
346)
347)     OFFSET EQU '2',  /* NUMBER OF TRACKS USED BY BOOT */
348)
349)     ALI EQU '0C0H', /* FIRST ALLOCATION
350)         VECTOR ELEMENT.  EACH BIT THAT IS '1' RESERVES
351)         A 1 K BLOCK FOR THE DIRECTORY.  EACH BLOCK IS
352)         8 RECORDS BY 128 BYTES PER RECORD  (NOTE THAT
353)         RESERVATIONS START ON THE LEFT OF THE WORD */
354)
355)     ALLOC0 (32) BYTE,  /* ALLOCATION VECTOR FOR DISK 0 */
356)     ALLOC1 (32) BYTE,  /* ALLOCATIONN VECTOR FOR DISK 1 */
357)     ALLOCA ADDRESS,    /* POINTER TO CURRENTLY REFERENCED ALLOC */
358)     ALLOC BASED ALLOCA (32) BYTE;  /* ALLOC VECTOR TEMPLATE */
359)
360)
```

```
361>    DECLARE
362>        EMP EQU '0E5H',
363>
364>        MRD EQU '10',  /* NUMBER OF READ RE-TRYS */
365>
366>        FOREVER EQU 'WHILE TRUE',
367>        MAL EQU '242', /* LARGEST BLOCK NUMBER */
368>        MRC  EQU '127', /* LARGEST RECORD NUMBER */
369>        DSF EQU '2',   /* AMOUNT TO SHIFT 128 BYTE RECORD
370>                          TO GET A SINGLE DISK ENTRY */
371>        DMK EQU '11B', /* MASK CORRESPONDING TO DSF */
372>        FLN EQU '32',
373>        FSL EQU '5', /* AMOUNT TO SHIFT TO MULTIPLY
374>                        BY THE FCB LENGTH (FLN) */
375>        FDM EQU '16',  /* BEGINNING OF DISK MAP */
376>        FRL EQU '32',   /* LOCATION OF REC TO R/W */
377>        FRC EQU '15',  /* LOCATION OF RECORD COUNT
378>                          (MUST BE ONE BELOW DISK MAP) */
379>        FRE EQU '12', /* POSITIONN OF REEL NUMBER */
380>        LFB EQU '31',
381>        FNM EQU '13'; /* LENGTH OF FILE NAME */
382>
383>        DECLARE S BASED INFO (33) BYTE; /* FILE CONTROL BLOCK
384>            PASSED TO THE DISK MONITOR FROM THE USER */
385>
386>
387>        /* THE FILE CONTROL BLOCK FORMAT IS SHOWN BELOW.
388>
389>           ------------------------------------------------------
390>           /  1 BY  /  8 BY  /  3 BY  /  1 BY  /2BY/1 BY/  16 BY    /
391>           /FILETYPE/  NAME  /  EXT   / REEL NO/XXX/RCNT/DM0 .. DM15/
392>           ------------------------------------------------------
393>
394>            FILETYPE :  0E5H IF AVAILABLE (OTHERWISE UNDEFINED NOW)
395>            NAME     :  8 CHARACTER PRIMARY NAME
396>            EXT      :  3 CHARACTER EXTENT
397>                           COM IMPLIES COMMAND TYPE
398>                           (OTHERWISE UNDEFINED NOW)
399>            REEL NO  :  'REEL NUMBER' FIRST REEL IS 0, SECOND IS 1,
400>                          AND SO FORTH UNTIL 255.
401>            XXX      :  UNUSED FOR NOW
402>            RCNT     :  RECORD COUNT IN FILE (0 TO 127)
403>            DM0 ...  :  DISK ALLOCATION MAP.  255 IF NOT ALLOCATED,
404>               DM15     OTHERWISE IT POINTS TO ALLOCATED DISK BLOCK
405>
406>            THE FILE CONTROL BLOCK IS FOLLOWED BY ONE BYTE OF
407>            INFORMATIONN WHICH GIVES THE NEXT RECORD TO BE READ
408>            OR WRITTEN INN AN OPENED FILE.  THIS INFORMATION
409>            IS NOT A PART OF THE DIRECTORY.  EACH READ OR WRITE
410>            WILL INCREMENT THIS RECORD COUNT.
411>
412>            */
413>
414>
415>    DECLARE
416>        OLDDSK BYTE,              /* DISK ON ENTRY TO DOS */
417>        FCBDSK BYTE,              /* DISK NAMED IN FCB */
418>        CURDSK BYTE INITIAL(0),   /* CURRENTLY ADDRESSED DISK */
419>        DLOG   BYTE INITIAL(0),    /* BIT VECTOR GIVING LOGGED-IN DISKS */
420>        CURTRKV(NDISK) BYTE,       /* TRACK VECTOR */
421>        CURRECV(NDISK) ADDRESS,    /* RECORD VECTOR */
422>        CURTRKA ADDRESS,           /* POINTS TO CURRENT TRACK NUMBER */
423>        CURRECA ADDRESS,           /* POINTS TO CURRENT RECORD NUMBER */
424>        CURREC BASED CURRECA ADDRESS, /* CURRENTLY ADDRESSED RECORD */
425>        CUPTRK BASED CURTRKA BYTE,  /* CURRENT TRACK 0-76 */
426>        RCOUNT BYTE,       /* RECORD COUNT IN CURRENTLY
427>                              ADDRESSED FCB */
428>        VRECORD BYTE,    /* CURRENT VIRTUAL RECORD */
429>        ARECORD ADDRESS;  /* CURRENT ACTUAL RECORD */
430>
431>
432>    PDISK: PROCEDURE;
433>        CALL PRINT('DISK $');
434>        CALL TABOUT('A'+CURDSK);
435>        END PDISK;
436>
437>NHOME: PROCEDURE;
438>     /* MOVE TO HOME POSITION, THEN OFFSET BY DOS TRACKS */
439>     CALL TRACK0;   /* AT HOME POSITION */
440>     CALL SELTRK(OFFSET);  /* SELECT FIRST DIRECTORY POSITION */
441>     CURREC, CURTRK = 0;
442>     END HOME;
443>
444>SEEK: PROCEDURE;
445>     /* SEEK THE TRACK GIVEN BY ARECORD (ACTUAL RECORD) */
446>     DECLARE TRAN. DATA /* SECTOR NUMBER TRANSLATE TABLE */
447>     (01H,07H,0DH,13H,  19H,05H,0BH,11H,  17H,03H,09H,0FH,
448>      15H,02H,08H,0EH,  14H,1AH,06H,0CH,  12H,18H,04H,0AH,
449>      10H,16H);
450>
451>     DECLARE T ADDRESS;
452>
453>         DO WHILE ARECORD < CURREC;
454>         CURREC = CURREC - 26;
455>         CURTRK = CURTRK - 1;
456>         END;
457>         DO WHILE ARECORD >= (T := CURREC + 26);
458>         CURREC = T;
459>         CURTRK = CURTRK + 1;
460>         END;
461>
462>     /* WE ARE NOW POSITIONED OVER THE TRACK CONTAINING THE ACTUAL
463>     RECORD.    THE SECTOR TO BE READ IS ARECORD - CURREC + 1.  THE
464>     TRACK NUMBER IS CURTRK */
465>     CALL SELTRK(CURTRK+OFFSET);
466>     CALL SELSEC(TRAN(ARECORD - CURREC));
467>     END SEEK;
468>
469>WAITIO: PROCEDURE(READING);
470>     DECLARE READING BYTE;  /* TRUE IF READING, FALSE IF WRITING */
471>     DECLARE COND BYTE;     /* CONDITION UPON, RETURN */
472>     DECLARE CTLC LITERALLY '03H';
473>     IF READING THEN COND = READ$DISK; ELSE
474>                     COND = WRITE$DISK;
475>     IF COND = 0 THEN RETURN;  /* DISK I/O SUCCESSFUL */
476>
477>     /* ARRIVE HERE AFTER TOO MANY READ WRITE FAILURES */
478>     CALL CRLF; CALL PRINT('PERM ERR $');
479>     CALL PDISK; IF CONIN = CTLC THEN GO TO BOOT;
480>     /* ENSURE NOT BATCH PROCESSING */
```

```
481)      IF (IOSTAT AND 11B) >  1 THEN HALT;
482)      CALL CRLF;
483)      END WAITIO;
484)
485)RDBUFF: PROCEDURE;
486)      /* START AN I/O AND WAIT FOR IO FINISH */
487)      CALL WAITIO(TRUE);
488)      END RDBUFF;
489)
490)WRBUFF: PROCEDURE;
491)      /* WRITE THE BUFFER, SELECT NON-DELETED DATA */
492)      CALL WAITIO(FALSE);
493)      END WRBUFF;
494)
495)
496)INDEX: PROCEDURE;
497)      /* COMPUTE DISK BLOCK NUMBER FROM CURRENT
498)         FCB ADDRESSED BY INFO */
499)
500)      ARECORD = $(FDM+SHR(VRECORD,3));
501)      END INDEX;
502)
503)AATRAN: PROCEDURE;
504)      /* COMPUTE ACTUAL TRACK ADDRESS (ASSUMES A
505)         PREVIOUS CALL TO INDEX */
506)      ARECORD = SHL(ARECORD,3) OR (VRECORD AND 111B);
507)      END ATRAN;
508)
509)GETFCB: PROCEDURE;
510)      /* SET VARIABLES FROM CURRENTLY ADDRESSED FCB */
511)
512)      VRECORD = $(FRL);
513)      RCOUNT  = $(FRC);
514)      END GETFCB;
515)
516)SETFCB: PROCEDURE;
517)      /* PLACE VALUES BACK INTO CURRENTLY ADDRESSED
518)      FCB, AND INCREMENT THE RECORD COUNT */
519)
520)      $(FRL) = VRECORD + 1;
521)      $(FRC) = RCOUNT;
522)      END SETFCB;
523)
524)SEEK$DIR: PROCEDURE;
525)      /* SEEK THE RECORD CONTAINING THE CURRENT DIRECTORY ENTRY */
526)      ARECORD = SHR(DCNT,DSF);
527)      CALL SEEK;
528)      END SEEK$DIR;
529)
530)READ$DIR: PROCEDURE;
531)      /* READ NEXT DIRECTORY ENTRY (SET DCNT=255 INITIALLY)*/
532)      IF (DCNT:=DCNT+1) >  DMX THEN
533)          DO; DCNT = 255; RETURN;
534)          END;
535)      IF (DPTR:=SHL(DCNT AND DMK,FSL)) = 0 THEN
536)          DO; CALL SEEK$DIR;
537)          CALL RDBUFF;
538)          END;
539)      END READ$DIR;
540)
```



```
541)GET$ALLOC: PROCEDURE(I) BYTE;
542)      DECLARE I BYTE;
543)      RETURN ALLOC(I);
544)      END GET$ALLOC;
545)
546)PPUT$ALLOC: PROCEDURE(I,X);
547)      DECLARE (I,X) BYTE;
548)      ALLOC(I) = X;
549)      END PUT$ALLOC;
550)
551)
552)GET$ALLOC$BIT: PROCEDURE(I) BYTE;
553)      /* RETURN THE I-TH BIT OF ALLOC */
554)      DECLARE I BYTE;
555)      RETURN ROL(ALLOC(SHR(I,3)), (I AND 111B)+1);
556)      END GET$ALLOC$BIT;
557)
558)SET$ALLOC$BIT: PROCEDURE(I,B);
559)      /* SET THE I-TH BIT OF ALLOC TO THE LSB OF B */
560)      DECLARE (I,B) BYTE;
561)      CALL PUT$ALLOC(SHR(I,3),
562)          ROR((GET$ALLOC$BIT(I) AND 0FEH) OR B, (I AND 111B) + 1));
563)      END SET$ALLOC$BIT;
564)
565)
566)GETBUFF: PROCEDURE(I) BYTE;
567)      DECLARE I BYTE;
568)      RETURNN BUFF(I);
569)      END GETBUFF;
570)
571)PPUTBUFF: PROCEDURE(I,X);
572)      DECLARE (I,X) BYTE;
573)      BUFF(I) = X;
574)      END PUTBUFF;
575)
576)SCANDM: PROCEDURE(BIT);
577)      DECLARE (BIT, I, K) BYTE;
578)      /* SCANDM SCANS THE DISK MAP ADDRESSED BY DPTR FOR NON-ZERO ENTRIES
579)      -- THE ALLOCATION VECTOR ENTRY CORRESPONDING TO A NON-ZERO ENTRY
580)      IS SET TO THE VALUE OF 'BIT' */
581)          DO I = DPTR+FDM TO DPTR+LFB;
582)          IF (K := GETBUFF(I)) <>  0 THEN
583)              CALL SET$ALLOC$BIT(K,BIT);
584)          END;
585)      END SCANDM;
586)
587)INITIALIZE: PROCEDURE;
588)      DECLARE I BYTE;
589)      /* INITIALIZE THE DISK SYSTEM */
590)      RET = FALSE; /* SET TO TRUE IF $ FILE EXISTS */
591)
592)      ALLOC = AL1;
593)          DO I=1 TO 31; CALL PUT$ALLOC(I,0);
594)          END;
595)      CALL HOME;
596)      DCNT = 255;
597)          DO FOREVER;
598)          CALL READ$DIR;
599)          IF DCNT = 255 THEN RETURN;
600)          IF GETBUFF(DPTR) <>  EMP THEN
```

```
601>            DO; /* CHECK FOR $ FILE (IN CASE OF SUBMIT) */
602>            RET = RET OR GETBUFF(DPTR+1) = '$';
603>            /* SET ALLOC BIT TO 1 FOR EACH NON-ZERO DM ENTRY */
604>            CALL SCANDM(1);
605>            END;
606>        END;
607>    END INITIALIZE;
608>
609>
610>DDECLARE SEARCHL BYTE,      /* SEARCH LENGTH SET BY SEARCH */
611>    SEARCHA ADDRESS;        /* SEARCH ADDRESS SET BY SEARCH */
612>
613>SEARCHN: PROCEDURE;
614>    /* SEARCH FOR THE NEXT DIRECTORY ELEMENT, ASSUMING A PREVIOUS
615>    CALL ON SEARCH WHICH SETS SEARCHA AND SEARCHL */
616>    DECLARE (I,C) BYTE;
617>    INFO = SEARCHA;
618>        DO FOREVER;
619>        CALL READ$DIR;
620>        IF (RET := DCNT) = 255 THEN RETURN;
621>        I = 0;
622>            DO WHILE (I < SEARCHL) AND
623>            /* MATCH OR QUESTIONN MARK */
624>            ((C := S(I)) = GETBUFF(DPTR+I) OR C = 63);
625>            I = I + 1;
626>            END;
627>        IF I = SEARCHL THEN RETURN;
628>        END;
629>    END SEARCHN;
630>
631>SEARCH: PROCEDURE(XL);
632>    DECLARE XL BYTE;
633>    SEARCHL = XL;
634>    SEARCHA = INFO;
635>    DCNT = 255;
636>    CALL HOME;
637>    /* NOW READY TO READ THE DISK */
638>    CALL SEARCHN;
639>    END SEARCH;
640>
641>NDELETE: PROCEDURE;
642>    DECLARE (I,J,K) BYTE;
643>    /* SEARCH ONLY UP THROUGH THREE CHARACTER EXTENT */
644>    CALL SEARCH(FRE);
645>      DO FOREVER;
646>      IF DCNT = 255 THEN /* NO MORE ENTRIES MATCH */ RETURN;
647>        /* SET EACH NON-ZERO DISK MAP ENTRY TO 0 IN ALLOC VECTOR */
648>        CALL SCANDM(0);
649>        CALL PUTBUFF(DPTR,EMP);
650>        /* ARECORD HAS BEEN PREVIOUSLY SOUGHT BY READDIR */
651>        CALL WRBUFF;
652>      CALL SEARCHN;
653>      END;
654>    END DELETE;
655>
656>GET$BLOCK: PROCEDURE(L) BYTE;
657>    /* FIND A BLOCK WHICH IS AVAILABLE ON THE DISK AND IS CLOSEST
658>    TO THE BLOCK 'L'.  RETURN A 0 IF NO BLOCK IS AVAILABLE */
659>    DECLARE (L, R) BYTE;
660>        R = L;
```



```
661>        DO WHILE (R < MAL) OR (L > 0);
662>        L = L - (1 AND L > 0);
663>        R = R + (1 AND R < MAL);
664>        IF NOT GET$ALLOC$BIT(R) THEN RETURN R;
665>        IF NOT GET$ALLOC$BIT(L) THENN RETURN L;
666>        END;
667>    RETURN 0;
668>    END GET$BLOCK;
669>
670>COPY$DIR: PROCEDURE(B,L);
671>    DECLARE (B,L) BYTE;
672>    /* COPY FCB INFORMATION STARTING AT BYTE B FOR L BYTES INTO
673>    BEGINNING OF CURRENTLY ADDRESSED DIRECTORY ENTRY */
674>        DO WHILE (L:=L-1) <> 255;
675>        CALL PUTBUFF(L+DPTR,S(B+L));
676>        END;
677>    CALL SEEK$DIR;
678>    CALL WRBUFF;
679>    END COPY$DIR;
680>
681>COPY$FCB: PROCEDURE;
682>    /* COPY THE ENTIRE FILE CONTROL BLOCK */
683>    CALL COPY$DIR(0,FRL);
684>    END COPY$FCB;
685>
686>RENAME: PROCEDURE;
687>    /* RENAME THE FILE DESCRIBED BY THE FIRST HALF OF THE CURRENTLY
688>    ADIRESSED FILE CONTROL BLOCK.  THE NEW NAME IS CONTAINED IN THE
689>    LAST HALF OF THE CURRENTLY ADDRESSED FILE CONTROL BLOCK.  THE
690>    FILE TYPE, FILE NAME, AND FILE EXT ARE CHANGED, BUT THE REEL
691>    NUMBER FIELD IS IGNORED */
692>
693>    /* SEARCH UP TO THE REEL NUMBER FIELD */
694>    CALL SEARCH(FRE); S(16)=S;
695>        DO WHILE DCNT <> 255; CALL COPY$DIR(FDN,FRE);
696>        CALL SEARCHN;
697>        END;
698>    END RENAME;
699>
700>OPEN: PROCEDURE;
701>    DECLARE I BYTE;
702>    /* SEARCH FOR DIRECTORY ENTRY, COPY TO FCB */
703>    CALL SEARCH(FNM);
704>    IF DCNT <> 255 THEN
705>        DO I=FNM TO LFB;
706>        S(I) = GETBUFF(DPTR+I);
707>        END;
708>    END OPEN;
709>
710>CLOSE: PROCEDURE;
711>    /* LOCATE THE DIRECTORY ELEMENT AND RE-WRITE */
712>    CALL SEARCH(FNM);
713>    IF DCNT <> 255 THEN
714>        CALL COPY$FCB;
715>    END CLOSE;
716>
717>MAKE: PROCEDURE;
718>    /* CREATE A NEW FILE, FIRST CREATE ENTRY IN
719>    THE DIRECTORY.  FILE IS OPENED UPON RETURN */
720>    DECLARE I BYTE,
```

```
721>          FCB ADDRESS;
722>
723>      FCB = INFO; INFO = .EMP;
724>      /* LOOK FOR AN EMPTY DIRECTORY ENTRY */
725>      CALL SEARCH(1);
726>      IF DCNT <>  255 THEN
727>          DO; /* SET ELEMENTS TO ZERO */
728>          INFO = FCB;
729>              DO I=FNM TO LFB;
730>              S(I) = 0;
731>              END;
732>          /* COPY INTO DIRECTORY ENTRY */
733>          CALL COPY$FCB;
734>          END;
735>      END MAKE;
736>
737>OPEN$REEL: PROCEDURE(READING);
738>      DECLARE READING BYTE;
739>      /* CLOSE CURRENT REEL AND OPEN THE NEXT ONE, IF POSSIBLE
740>      READING IS TRUE IF WE ARE IN READ MODE */
741>      CALL CLOSE;
742>      /* RET REMAINS AT 255 IF WE CANNOT OPEN THE NEXT REEL */
743>      IF DCNT = 255 THEN. RETURN;
744>      /* INCREMENT THE REEL NUMBER */
745>      S(CFRE) = S(CFRE) + 1;
746>      CALL SEARCH(FNM);
747>      IF DCNT = 255 THEN
748>          DO; IF READING THEN RETURN;
749>          CALL MAKE;
750>          END; ELSE
751>      CALL OPEN;
752>      IF DCNT = 255 THEN
753>          DO; RET = 1; /* END OF FILE IN DISK READ */
754>          RETURN;
755>          END;
756>      CALL GETFCB;
757>      RET = 0;
758>      END OPEN$REEL;
759>
760>NDISKREAD: PROCEDURE;
761>      CALL GETFCB;
762>
763>      IF RCOUNT <= VRECORD THEN
764>          DO; RET = 1;
765>          IF VRECORD = 128 THENN CALL OPEN$REEL(TRUE);
766>          VRECORD = 0;
767>          IF RET <>  0 THEN RETURN;
768>          END;
769>          DO; CALL INDEX;
770>
771>          /* ERROR 2 IF READING UNWRITTTEN DATA */
772>          IF LOW(ARECORD) = 0 THEN RET = 1; ELSE
773>              DO;  CALL ATRAN;
774>              /* ARECORD IS NOW ACTUAL DISK ADDRESS */
775>              CALL SEEK;
776>              /* NOW READ THE BUFFER */
777>              CALL RDBUFF;
778>              CALL SETFCB;
779>              END;
780>          END;
```



```
781>      END DISKREAD;
782>
783>DDISKWRITE: PROCEDURE;
784>      DECLARE (I,L) BYTE;
785>      CALL GETFCB;
786>
787>      IF VRECORD >  MRC THEN /* PAST EOF, NEXT REEL NOT OPENED */
788>          RET = 1; ELSE
789>          DO; CALL INDEX;
790>          IF LOW(ARECORD) = 0 THEN. /* NOT ALLOCATED */
791>              DO; /* THE ARGUMENT TO GET$BLOCK IS THE STARTING POSITION
792>              FOR THE DISK SEARCH - THIS SHOULD BE THE LAST ALLOCATED
793>              BLOCK FOR THIS FILE, OR THE VALUE 0 IF NO SPACE HAS BEEN
794>              ALLOCATED TO THIS FILE */
795>              I = 0;
796>              IF (L := FDM + SHR(VRECORD,3)) >  FDM THEN
797>              /* THERE IS A PREVIOUS BLOCK ALLOCATED */ I = S(L-1);
798>              IF (I := GET$BLOCK(I)) = 0 THEN /* NO MORE SPACE */
799>                  RET = 2; ELSE
800>                  DO; CALL SET$ALLOC$BIT(I,1);
801>                  /* BLOCK IS ALLOCATED */
802>                  ARECORD. S(L) = I;
803>                  END;
804>              END;
805>          /* CONTINUE IF NO ERROR IN ALLOCATION */
806>          IF RET = 0 THEN
807>              DO; CALL ATRAN;
808>              CALL SEEK;
809>              CALL WRBUFF;
810>              IF RCOUNT <= VRECORD THEN RCOUNT = VRECORD+1;
811>              /* CHECK FOR END-OF-REEL, IF FOUND ATTEMPT TO OPEN
812>              NEXT REEL IN PREPARATION FOR THE NEXT WRITE  */
813>              IF VRECORD = MRC THEN
814>                  DO;
815>                  /* UPDATE CURRENT FCB BEFORE GOING TO THE NEXT REEL */
816>                  CALL SETFCB; CALL OPENREEL(FALSE);
817>                  /* VRECORD REMAINS AT MRC CAUSING END-OF-FILE
818>                  IF NO MORE DIRECTORY SPACE IS AVAILABLE */
819>                  IF RET = 0 THEN VRECORD = 255; /* GOES TO ZERO */
820>                  RET = 0;
821>                  END;
822>              CALL SETFCB;
823>              END;
824>          END;
825>      END DISKWRITE;
826>
827>SELECT: PROCEDURE;
828>      /* SELECT DISK 'INFO' FOR SUBSEQUENT
829>      INPUT OR OUTPUT OPERATIONS */
830>
831>      IF CURDSK >  MAXDSK THEN. /* SELECTIONN ERROR */
832>          DO; CALL CRLF; CALL PRINT('SELECT ERROR $');
833>          CALL PDISK; CALL CRLF; GO TO BOOT;
834>          END;
835>      ALLOCA = .ALLOC0(SHL(CURDSK,5));
836>      /* NOTE THAT THIS ASSUMES THERE ARE NO MORE
837>      THAN 8 DISKS ON THE SYSTEM - OTHERWISE
838>      REPLACE BY .ALLOC0(SHL(DOUBLE(CURDSK),5)) */
839>
840>      CURTRKA = .CURTRKV(CURDSK);
```

```
841>      CURRECA = .CURRECV(CURDSK);
842>
843>      /* SET CONTROLLER */
844>      CALL SELDISK(CURDSK);
845>
846>      /* CHECK TO INSURE THAT DISK IS LOGGED IN */
847>      IF NOT ROR(ROL(DLOG.1),CURDSK+1) THEN
848>          DO;
849>          DLOG = DLOG OR ROR(ROL(1,CURDSK+1),1);
850>          CALL INITIALIZE;
851>          END;
852>      END SELECT;
853>
854>CURSELECT: PROCEDURE;
855>      IF LINFO <>  CURDSK THEN
856>          DO; CURDSK = LINFO; CALL SELECT;
857>          END;
858>      END CURSELECT;
859>
860>RESELECT: PROCEDURE;
861>      /* CHECK CURRENT FCB TO SEE IF RESELECTION NECESSARY */
862>      IF (LINFO := (S AND 1$1111B) -1) < 30 THEN
863>          DO; OLDDSK = CURDSK; FCBDSK = S; S = S AND 1110$0000B;
864>          CALL CURSELECT;
865>          END;
866>      END RESELECT;
867>
868>SETDMA: PROCEDURE(A);
869>      DECLARE A ADDRESS;
870>      CALL SELDMA(BUFFA.*A);
871>      END SETDMA;
872>
873>  /* ARRIVE HERE UPON ENTRY TO THE DISK MONITOR.
874>  SAVE THE STACKPOINTER, PERFORM THE DESIRED FUNCTION,
875>  RESTORE THE STACKPOINTER, AND RETURN TO THE CALLING
876>  PROGRAM. */
877>  DECLARE STACK (16) ADDRESS,
878>      OLDSP ADDRESS;
879>
880>  OLDSP = STACKPTR;
881>  STACKPTR = .STACK(LENGTH(STACK));
882>  /* CALLING PROGRAM'S STACK TOP ADDRESS NOW SAVED */
883>
884>  LINFO = LOW(INFO);
885>  ARET, RET = 0;
886>    FCBDSK = 0;
887>
888>    DO CASE FUNC;
889>    /* 0: SYSTEM RE-BOOT */
890>    GO TO BOOT;
891>    /* 1: READ CONSOLE */
892>        DO; /* READ CHARACTER, TEST FOR GRAPHICS */
893>        IF ((RET := CONIN) >= ' ') OR
894>            (RET = CR) OR (RET = LF) OR (RET = TAB) THEN
895>            CALL TABOUT(RET);
896>        END;
897>    /* 2: WRITE CONSOLE */
898>    CALL TABOUT(LINFO);
899>    /* 3: READ READER DEVICE */
900>    RET = READIN;
```



```
901>    /* 4: WRITE PUNCH DEVICE */
902>    CALL PUNOUT(LINFO);
903>    /* 5: WRITE LIST DEVICE */
904>    CALL LSTOUT(LINFO);
905>    /* 6: INTERROGATE MEMORY SIZE */
906>    ARET = FDOS;
907>    /* 7: INTERROGATE DEVICE STATUS */
908>    ARET = IOSTAT;
909>    /* 8: CHANGE DEVICE STATUS */
910>    IOSTAT = INFO;
911>    /* 9: PRINT BUFFER AT THE CONSOLE  */
912>    CALL PRINT(INFO);
913>    /* 10: READ BUFFER FROM THE CONSOLE */
914>    CALL READ;
915>    /* 11: CHECK FOR CONSOLE INPUT READY */
916>    RET = CONBRK;
917>    /* 12: */
918>    ;
919>    /* 13: RESET DISK SYSTEM, INITIALIZE TO DISK 0 */
920>        DO; CURDSK,DLOG = 0;
921>        CALL SETDMA(80H);
922>        CALL SELECT;
923>        CHAR$RDY,LISTCOPY = FALSE;
924>        END;
925>    /* 14: SELECT DISK 'INFO' */
926>        CALL CURSELECT;
927>    /* 15: OPEN */
928>        DO; CALL RESELECT;
929>        CALL OPEN;
930>        END;
931>    /* 16: CLOSE */
932>        DO; CALL RESELECT;
933>        CALL CLOSE;
934>        END;
935>    /* 17: SEARCH FOR FIRST OCCURRENCE OF A FILE */
936>        DO; CALL RESELECT;
937>        CALL SEARCH(FNM);
938>        END;
939>    /* 18: SEARCH FOR NEXT OCCURRENCE OF A FILE NAME */
940>        DO; INFO = SEARCHA; CALL RESELECT;
941>        CALL SEARCHN;
942>        END;
943>    /* 19: DELETE A FILE */
944>        DO; CALL RESELECT;
945>        CALL DELETE;
946>        END;
947>    /* 20: READ A FILE */
948>        DO; CALL RESELECT;
949>        CALL DISKREAD;
950>        END;
951>    /* 21: WRITE A FILE */
952>        DO; CALL RESELECT;
953>        CALL DISKWRITE;
954>        END;
955>    /* 22: CREATE A FILE */
956>        DO; CALL RESELECT;
957>        CALL MAKE;
958>        END;
959>    /* 23: RENAME A FILE */
960>        DO; CALL RESELECT;
```

```
961>          CALL RENAME;
962>          END;
963>      /* 24: RETURN THE LOGIN VECTOR */
964>      RET = DLOG;
965>      /* 25: RETURN SELECTED DISK NUMBER */
966>      RET = CURDSK;
967>      /* 26: SET THE SUBSEQUENT DMA ADDRESS TO INFO */
968>      CALL SETDMA(INFO);
969>      /* 27: RETURN THE LOGIN VECTOR ADDRESS  */
970>      ARET = ALLOCA;
971>      END; /* OF CASES */
972>
973>
974>
975>  GOBACK:
976>    IF FCBDSK <>  0 THEN  /* RESTORE DISK NUMBER */
977>        DO; S = FCBDSK; LINFO = OLDDSK; CALL CURSELECT;
978>        END;
979>  /* RESTORE THE USER'S STACK AREA */
980>  STACKPTR = OLDSP;
981>
982>  /* RETURN. A SINGLE OR DOUBLE BYTE VALUE */
983>  RETURNN ARET OR RET;
984>  END DISKMON;
985>  /* <><><><><><><><><>   END OF DISK MONITOR <><><>  */
986>
987>;
988>NDECLARE TEMPDATA(10) BYTE;  /* TEN LOCATIONS IN. RAM FOR THE INTERFACE*/
989>EOF
990>EOF;
```

```
  1>
  2>       5 MEMORY 03E00H
  3>      25 DISKMON 03206H
  4>      26 FUNC 03D33H
  5>      27 INFO 03D34H
  6>      30 COPYRIGHT 03211H
  7>      32 LINFO 03D37H
  8>      33 ARET 03D38H
  9>      34 RET 03D3AH
 10>      35 CHAPRDY 03D3CH
 11>      37 K2CHAR 03D3DH
 12>      38 LISTCOPY 03D3EH
 13>      39 CONRDY 03235H
 14>      42 CONINN 03235H
 15>      46 CONBRK 0324AH
 16>      56 CONCHAR 0327DH
 17>      57 CHAR 0327FH
 18>      60 CONOUT 03285H
 19>      61 CHAP 03D40H
 20>      64 LSTOUT 03298H
 21>      65 CHAP 03D41H
 22>      68 PUNOUT 032A0H
 23>      69 CHAP 03D42H
 24>      72 READIN 032A8H
 25>      75 TRACK8 032ACH
 26>      78 SELDISK 032C0H
 27>      79 DISK 03D43H
 28>      82 SELTRK 032D8H
 29>      83 TRACK 03D44H
 30>      86 SELSEC 032E0H
```



```
 31>      87 SECTOR 03D45H
 32>      90 SELDMA 032CCH
 33>      91 DMA 03D46H
 34>      94 READDISK 032D2H
 35>      97 WRITEDISK 032D6H
 36>     100 COLUMN 03D45H
 37>     101 IOSTATA 03D4AH
 38>     103 TABOUT 032DAH
 39>     104 CHAR 03D4CH
 40>     106 I 03D4DH
 41>     110 CRLF 03335H
 42>     121 PRINT 03340H
 43>     122 A 03D4EH
 44>     124 I 03D51H
 45>     129 READ 03362H
 46>     132 COMLEN 03D52H
 47>     134 C 03D53H
 48>     135 CTLOUT 03365H
 49>     164 DPTR 03D54H
 50>     165 DCNT 03D55H
 51>     166 BUFFA 03D56H
 52>     169 ALLOC0 03D58H
 53>     170 ALLOC1 03D78H
 54>     171 ALLOCA 03D98H
 55>     174 OLDDSK 03D9AH
 56>     175 FCBDSK 03D9BH
 57>     176 CURDSK 03D9CH
 58>     177 DLOG 03D9DH
 59>     178 CURTRKV 03D9EH
 60>     179 CURRECV 03DA0H
 61>     180 CURTRKA 03DA4H
 62>     181 CURRECA 03DA6H
 63>     184 RCOUNT 03DA8H
 64>     185 VRECORD 03DA9H
 65>     186 ARECORD 03DAAH
 66>     187 FDISK 0343AH
 67>     192 HOME 03454H
 68>     194 SEEK 0345AH
 69>     197 TRAN 03462H
 70>     213 T 03DACH
 71>     218 WAITIO 03508H
 72>     219 READING 03DAEH
 73>     221 COND 03DAFH
 74>     229 RDBUFF 03560H
 75>     231 WRBUFF 03566H
 76>     233 INDEX 0356CH
 77>     235 ATRAHN 03580H
 78>     237 GETFCB 035A8H
 79>     240 SETFCB 035C1H
 80>     242 SEEKDIR 035DFH
 81>     244 READDIR 035F1H
 82>     250 GETALLOC 03614H
 83>     251 I 03DB0H
 84>     253 PUTALLOC 03620H
 85>     254 I 03DB1H
 86>     255 X 03DB2H
 87>     257 GETALLOCBIT 03636H
 88>     258 I 03DB3H
 89>     260 SETALLOCBIT 03659H
 90>     261 I 03DB4H
```

```
 91>     262 B 03DB5H
 92>     264 GETBUFF 03686H
 93>     265 I 03DB6H
 94>     267 PUTBUFF 03692H
 95>     268 I 03DB7H
 96>     269 X 03DB8H
 97>     271 SCANDM 036A8H
 98>     272 BIT 03DB9H
 99>     274 I 03DBAH
100>     275 K 03DBBH
101>     278 INITIALIZE 036D8H
102>     282 I 03DBCH
103>     291 SEARCHL 03D9DH
104>     292 SEARCHA 03DBEH
105>     293 SEARCHN 03736H
106>     295 I 03DC0H
107>     296 C 03DC1H
108>     303 SEARCH 0379BH
109>     304 XL 03DC2H
110>     306 DELETE 037B7H
111>     308 I 03DC3H
112>     309 J 03DC4H
113>     310 K 03DC5H
114>     314 GETBLOCK 037DBH
115>     315 L 03DC6H
116>     317 R 03DC7H
117>     323 COPYDIR 03925H
118>     324 B 03DC8H
119>     325 L 03DC9H
120>     329 COPYFCB 03855H
121>     331 RENAME 0385DH
122>     335 OPEN 03879H
123>     337 I 03DCAH
124>     341 CLOSE 03882H
125>     344 MAKE 038C1H
126>     346 I 03DCBH
127>     347 FCB 03DCCH
128>     352 OPENREEL 03910H
129>     353 REWIND 03DCFH
130>     360 DISKREAD 03963H
131>     367 DISKWRITE 039A7H
132>     369 I 03DD0H
133>     370 L 03DD1H
134>     381 SELECT 03A5DH
135>     387 CUPSELECT 03ADEH
136>     390 RESELECT 03AF2H
137>     395 SETDMA 03B18H
138>     396 A 03ID2H
139>     398 STACK 03DD4H
140>     399 DLDSP 03DF4H
141>     432 GOBACK 03CF1H
142>     434 TEMPDATA 03DF6H
```

CP/M VERSION _____

SER. # _____