CHAPTER 13

PDOS UTILITIES

A PDOS utility is an auxiliary program that resides on the disk. Written in either assembly language or BASIC, PDOS utilities are run by simply entering the name of the desired utility. Of course, the utility must be a file on the disk with the appropriate attributes. The sections that follow describe each utility, the language in which it is written, its format and memory requirements, and the various PDOS systems on which it runs. Then a detailed description is given along with an example of the utility's output.

(CHAPTER 13 - PDOS UTILITIES continued)

13.1 ALOAD

Name: ALOAD
Function: Object Loader
Format: ALOAD {file1,file2,...}

Systems: All
Language: Assembly
Memory: 2K bytes + Modules

Restrictions: The ALOAD utility removes memory from the user's memory space.

Description:

The ALOAD utility loads object code modules into user task memory. ALOAD loads both absolute and relocatable object code. However, the primary objective is to load relocatable object code with a few absolute patches such as interrupt or XOP vectors. Therefore, all absolute location tags (tag 9) are output to the user console as a public service to the user as they are encountered. The absolute locations are loaded at that time.

No task entry is made in the execution queue and the user task memory is reduced by the total module sizes rounded to the next 1K byte boundary.

ALOAD first prints the current task memory limits. Next, up to 8 object files can be loaded. As each one is loaded, IDT labels and any absolute addresses are listed to the user console. When all the files have been loaded, ALOAD asks for an output file for the load map. The default output is the user console. The load map shows the names of the files loaded along with their IDT labels, memory limits, and entry addresses. Note: the loaded modules are called by their entry address.

Finally, the new task limits are listed. Allocation is in 1K bytes increments. To avoid unnecessary fracturing of memory, the assembly programs are allocated memory beginning at the low address of the task memory block. This is different from a CREATE task command. ALOAD is used for permanent allocation while the CREATE task is a temporary allocation and is recovered when the new task is terminated.

All permanent programs should be loaded at the same time for best memory usage. The load map should be saved for entry address reference. A small utility could be written to automatically retrieve the start addresses from the memory map.

The following is an example of how the ALOAD utility could be used in conjunction with a PDOS BASIC program. The assembly program COUNT:SR is listed to the right.

```
.SF COUNT:SR
*       COUNT:SR
*
        RORG 0
        IDT 'COUNT'
*
CNT     MOV @2(7),R1    ;COM[0]
        MOV @8(7),R2    ;COM[1]
        XBUG 1          ;**DEBUG PRINT**
*
CNT2    CLR *R2+        ;FILL ARRAY
        MOV R1,*R2+     ;  WITH DESCENDING #'S
        CLR *R2+
        DEC R1          ;DONE?
          JGT CNT2      ;N
        XPMC            ;Y
          DATA MES1
        RT
*
MES1    BYTE >0A,>0D
        TEXT 'IT WORKS'
        BYTE 0
        END CNT
```

(13.1 ALOAD continued)

Examples:

```
.LT
TASK  PAGE  TIME    SB    WS    PC    SR    BM    EM   CRU   PORT

*0     0     3    >6020 >619A >0762 >1005 >6000 >E000 >0080 >0001
.ALOAD
ASSEMBLY OBJECT LOADER R2.4
CURRENT TASK MEMORY LIMITS:
      LOW = >6000
     HIGH = >E000
FILE NAME=COUNT:SR
 *IDT=COUNT
FILE NAME=XBUG
 *IDT=XBUG2.4
 *ABS ADR=>0070                                          XOP vector 12
FILE NAME=<cr>
OUTPUT=<cr>
FILE NAME        IDT       ENTRY  LOW   HIGH

COUNT:SR         COUNT     >6000 >6000 >6026              COUNT called at address >6000
XBUG             XBUG2.4   >6046 >6026 >60C5

NEW TASK MEMORY LIMITS:
      LOW = >6400
     HIGH = >E000
.LT
TASK  PAGE  TIME    SB    WS    PC    SR    BM    EM   CRU   PORT

*0     0     3    >6420 >659A >0762 >1005 >6400 >E000 >0080 >0001
.EX
*READY
LOAD "TEST"
*READY
LIST
 10  DIM A[20]
 20  COM[0]=15
 30  COM[1]=ADR[A[0]]
 40  CALL #06000H
 50  FOR I=0 TO 20
 60   PRINT A[I];
 70  NEXT I
RUN

>PC=600A  WS=6400  SR=9005
 R0=0000 R1=000F R2=DE60 R3=5956 R4=009C R5=DEEC R6=6E3A R7=DEE2
 R8=670A R9=6420 RA=6750 RB=473C RC=6402 RD=66AC RE=6750 RF=DEDE
IT WORKS 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 0 0 0 0 0
STOP AT 70
```

13.2 ASM

Name: ASM
Function: PDOS 9900 Assembler
Format: ASM
ASM source,{object},{list},{error},{xref}

Systems: All
Language: Assembly
Memory: 10K bytes minimum

Restrictions: none

Description:

The ASM utility is the general purpose TMS 9900 and TMS 9995 assembler which is a major utility for developing assembly language programs. See Chapter 11, EDIT, ASM, LINK, XBUG, for a complete description of ASM. The following information from Chapter 11 is listed here for convenience.

The current errors are defined as:

'B' = Byte overflow
'C' = Illegal ASCII constant
'D' = Text delimiter error
'E' = Illegal number
'F' = File error
'M' = Multiply-defined symbol
'N' = Numeric overflow
'O' = Field overflow
'R' = CRU displacement out of range
'S' = Illegal symbol
'T' = Symbol table overflow
'U' = Undefined symbol
'X' = Missing operand
'e' = Expression mode error
'f' = Floating point conversion error
'm' = Multiply-defined symbol referenced
't' = Truncation error

The binary operators interpreted by ASM for expressions are:

+	addition
-	subtraction
*	signed multiplication
/	signed division
&	logical AND
!	logical inclusive OR
\	logical shift right (- shift left)

ASM source,{object},{list},{error},{xref}

source	Assembly source file
object	TI object output file
list	Assembly listing file
error	Assembly error file
xref	Cross reference file

Symbol types:

A = absolute
R = program-relocatable
D = data-relocatable
U = undefined
M = multiply-defined
E = REF symbol

(13.2 ASM continued)

The PDOS assembler supports the following directives:

```
IDT ('name')              Program identifier
END (exp)                 End assembly, set entry
LINK (file)               Link to file

EQU (exp)                 Define assembly-time constant
BYTE (exp1){,(exp2)}...   Initialize byte
DATA (exp1){,(exp2)}...   Initialize word
TEXT {-}{+}..('string')   Initialize text (any delimiter)
CONS (fp #){,(fp #)}...   Initialize 6 byte FP number

AORG (exp)                Absolute origin
RORG {(exp)}              Relocatable origin
DORG (exp)                Dummy origin (no object)
PSEG                      Program segment
DSEG                      Data segment

BES (exp)                 Block ending with symbol
BSS (exp)                 Block starting with symbol
EVEN                      Set word boundary

PAGE                      Page eject
TITL ('string')           Page title
LIST                      List source
UNL                       No source list
DXOP (symbol),(exp)       Define extended operation

DEF (sym){,(sym)}...      External definition
REF (sym){,(sym)}...      External reference
COPY (file)               Include from (file)

IFZ (exp),(symbol)        If (exp) zero, goto (symbol)
IFN (exp),(symbol)        If (exp) non-zero, goto (symbol)
DUP (exp)                 Duplicate next line (exp) times
OPT (char)=(exp){,(char)=(exp)}...
                          Set option flag (char)
        ?       QFLG      Assemble if ? non-zero
        #       PFLG      Assemble right or left half
        C       CFLG      Output checksum records
        L       LFLG      Expanded list options
        R       RFLG      Register cross reference
        S       SFLG      Punch symbol table to object
        X       XFLG      Output XREF to LIST file
```

(13.2 ASM continued)

The assembler outputs the following object tags:

TAG	FIELD 1	FIELD 2	FUNCTION	LIST ID
0	PSEG length	Program ID (8)	IDT	
1	Absolute entry	(not used)	END absolute	
2	Relocatable entry	(not used)	END relocatable	
3	Symbol (6)	(not used)	External REF	+
4	Absolute value	Symbol (6)	External DEF	
5	P-R value	Symbol (6)	P-R External DEF	
6	D-R value	Symbol (6)	D-R External DEF	
7	(Reserved)			
8	(Reserved)			
9	Absolute Address	(not used)	AORG	
A	P-R Address	(not used)	RORG,PSEG	
B	Absolute Data	(not used)	Absolute	blank
C	P-R Data	(not used)	Program-relocatable	'
D	(Reserved)			
E	(Reserved)			
F	(not used)	(not used)	End of record	
G	D-R Address	(not used)	RORG,DSEG	
H	D-R Data	(not used)	Data-relocatable	"

Examples: See Chapter 11.

13.3 BACKUP

Name: BACKUP
Function: Disk backup or copy
Format: BACKUP

Systems: All
Language: Assembly
Memory: 4K bytes minimum

Restrictions: This utility destroys all data on the destination disk.

NOTE: Upon receipt of the PDOS package, the PDOS system disk should be copied to another disk, the original stored as a master, and the copy used for actual system operation.

Description:

The BACKUP utility performs a sector by sector disk copy using one or two disk drives.

BACKUP first asks for the source disk number and the destination disk number. If only one drive is in the system, then enter the same number for both. The original and backup disks are swapped in and out until the entire source disk is copied. If two drives are in the system, then be sure to put the original in the drive corresponding to the source disk number.

Next, the program prompts for the number of sectors to be copied. For a TM990/303A 8" double density system, the total number of sectors to copy is 1976, which is the <CR> default value. BACKUP then prints a 'READY' prompt while you insert the original in the specified source drive and, if using two drives, insert the target disk in the destination drive. When you are ready, type 'Y'.

The disk name from the source disk is read, displayed, and you are asked to verify the transfer. If you enter a 'Y', the disk duplication begins. As many sectors as possible are read into the task's memory and then written to the new disk. As each block is completed, the number of the last sector copied is printed. Disk swapping prompts are output if only one drive is used.

Finally, since the BACKUP process also duplicates the name of the source disk on to the destination disk, the user is prompted for a new name for the destination disk. If only a carriage return if entered, the destination disk retains the same name as the source disk.

(13.3 BACKUP continued)

Examples:

```
.BACKUP
DISK BACKUP R2.4
SOURCE DISK #=1
DESTINATION DISK #=1
NUMBER OF SECTORS=1976
READY?Y
DUPLICATE 'PDOS 2.4 #1'?Y

INSERT SOURCE DISK IN DRIVE 1.  HIT <CR>....<cr>
INSERT DESTINATION DISK IN DRIVE 1.  HIT <CR>....<cr>
FINISHED SECTOR 122
INSERT SOURCE DISK IN DRIVE 1.  HIT <CR>....<cr>
INSERT DESTINATION DISK IN DRIVE 1.  HIT <CR>....<cr>
WRITE ERROR 16194 AT SECTOR 140.  RETRY?Y
...
FINISHED SECTOR 1975
SUCCESS!!  BACKUP NAME = PDOSBACKUP
RENAMED 'PDOSBACKUP'
._
```

13.4 BFIX

Name: BFIX
Function: Customize/write PDOS boot
Format: BFIX

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: The BFIX can only be run by task 0 and should only be run right after the system has been booted.

Description:

The BFIX utility customizes the PDOS system to your hardware configuration. Many system parameters are read from the currently running PDOS system and you are prompted for any desired changes. The clear screen and position cursor control sequences can be altered either by selecting one of the terminal types, whose function sequences are already defined, or by inputting the desired characters for each function, as prompted. All the BASIC screen functions can be changed in the same way.

The intervals of the four systems timer events, the system clock interrupt interval, the initial port CRU base addresses, and the interrupt enable mask can all be altered by BFIX. BFIX also reads the auto-start flag, the auto-start file name, the initial memory sizing (zeroing) limit, and the PDOS prompt characters for you to change.

Finally, the destination disk number and sector for the new system to be written to is input. A table of default boot sector numbers for various disk types is printed as a help. BFIX adjusts calculates a new memory checksum and then memory for >0000 to >5FFF is written to the designated disk starting at the specified boot sector number.

It is not necessary to output the new system to a disk. BFIX can be used to customize the system just temporarily.

(13.4 BFIX continued)

Examples:

The following is the output from a sample run of BFIX which changes nothing in the system except the PDOS prompt characters. No boot was written to disk.

```
.BFIX                                                    Invoke BFIX
BOOT FIX R2.4                                            BFIX banner
**CAUTION: EXECUTE ONLY AFTER NEW BOOT!
TERMINALS:                                               Predefined terminals
    A=ADDS REGENT 25
    D=DECSCOPE (VT52 or VT100)
    H=HAZELTINE 1520
    I=INTERTUBE II
    L=LEAR SEIGLER
    S=SOROC
    U=USER DEFINED
TYPE=U                                                   Do user definition sequence
CLEAR SCREEN CHARACTERS.....<ESC>*     NEW=<cr>          Examine SOROC function codes
POSITION CURSOR LEAD CHARACTERS..<ESC>=    NEW=<cr>      already in system, but don't
BASIC SCREEN CONTROL TABLE:                              alter them (just <CR>)
   (U) CURSOR UP.............<0B>       NEW=<cr>
   (D) CURSOR DOWN...........<LF>       NEW=<cr>
   (R) CURSOR RIGHT..........<FF>       NEW=<cr>
   (L) CURSOR LEFT...........<BS>       NEW=<cr>
   (B) BEGINNING OF LINE.....<CR>       NEW=<cr>
   (H) HOME CURSOR...........<1E>       NEW=<cr>
   (S) CLEAR TO END OF SCREEN....<ESC>Y NEW=<cr>
   (E) CLEAR TO END OF LINE......<ESC>T NEW=<cr>
   (W) RESET WRITE PROTECT.......<ESC>G NEW=<cr>
   (P) SET WRITE PROTECT.....<ESC>F     NEW=<cr>
   (() START WRITE PROTECT.....<ESC>I   NEW=<cr>
   ()) END WRITE PROTECT.....<ESC>H     NEW=<cr>
   (Z) CLEAR UNPROTECTED.....<ESC>K     NEW=<cr>
   (N) SKIP TO NEXT FIELD.....<09>      NEW=<cr>
ADJUST TIMER EVENTS (Y OR N)?Y                           Let's look at the timer intervals
```

(13.4 BFIX continued)

```
EVENT 112 = 25 TICS.  NEW TIC COUNT=<cr>                One is in TIC units
EVENT 113 = 1 SECONDS.  NEW SECOND COUNT=<cr>           # are in seconds
EVENT 114 = 10 SECONDS.  NEW SECOND COUNT=<cr>
EVENT 115 = 20 SECONDS.  NEW SECOND COUNT=<cr>
ADJUST SYSTEM CLOCK (Y OR N)?Y                          We could change the number
CURRENT TICS/SECONDS=125                                of TICs per seconds
    1=375 or 2.666 ms
    2=125 or 8 ms
    3=75 or 13.333 ms
    4=25 or 40 ms
NEW TICS/SECOND OPTION (1-4)=<cr>                       But let's not
ADJUST CONSOLE CRU BASE ADDRESSES (Y OR N)?Y            Here are the 9902 CRU bases
PORT #1 = >0080.  NEW HEX BASE=<cr>
PORT #2 = >0180.  NEW HEX BASE=<cr>
PORT #3 = >0E00.  NEW HEX BASE=<cr>
PORT #4 = >0A00.  NEW HEX BASE=<cr>
PORT #5 = >0A40.  NEW HEX BASE=<cr>
PORT #6 = >0A80.  NEW HEX BASE=<cr>
PORT #7 = >0AC0.  NEW HEX BASE=<cr>
PORT #8 = >0B00.  NEW HEX BASE=<cr>
CHANGE SYSTEM INTERRUPT MASK (Y OR N)?Y
CURRENT ENABLED LEVELS=3,4,5,6                          Clock and 9902 levels only
     ENTER NEW LEVELS=<cr>
AUTO-START UPON BOOT (Y OR N)?N                         This sets byte addr >0070 non-zero
ADJUST AUTO-START FILE NAME (Y OR N)?Y                  Name your auto-start file
CURRENT FILE NAME=S$STRT                                after yourself
   NEW FILE NAME=<cr>
ADJUST INITIAL MEMORY LIMIT (Y OR N)?Y                  In case there is need to protect a part
CURRENT MEMORY LIMIT=E000                               of high memory, lower this address
   NEW MEMORY LIMIT=<cr>
ADJUST PDOS PROMPT (Y OR N)?Y                           Don't like bells? Try angles!
  CURRENT PROMPT CHARACTERS......<BEL>.      NEW=>>
DISK #=<esc>                                            Hit escape for no write
>>_                                                     See new prompt characters!
```

13.5 BURN302

Name: BURN302
Function: Burn EPROMs with TM990/302
Format: BURN302

Systems: 101,102 only
Language: Assembly
Memory: 4k bytes + buffer

Restrictions: Only usable in TM990 systems.

Description:

The BURN302 utility allows programs and data to be programmed into EPROMs using the TM990/302 development module. BURN302 prompts with an '*', after which commands can be entered. This includes the loading of TI9900 tag object files, binary files, and data from an EPROM. Also, the utility buffer can be examined and changed. If a carriage return is entered the following command summary is printed, along with the current buffer limits:

```
*<cr>
HIGHEST PC=0000
BUFFER LIMITS ARE 0000 TO 7074
 0,<file>,<adr>           LOAD BINARY FILE
 1,<file>{,<adr>}         LOAD OBJECT FILE
 2,<adr1>,<adr2>,<byte>   LOAD EPROM DATA
 A                        VERIFY BLANK EPROM
 B{,<adr>}                SET BUFFER BASE
 C,<adr1>,<adr2>,{<adr3>} COMPUTE CHECKSUM
 E                        EXIT TO PDOS
 I{,<adr>}                SET EPROM INDEX
 M                        MODIFY BUFFER MEMORY
   <adr1>                 INSPECT
   <adr1>,<adr2>          DISPLAY MEMORY
   <adr1>,<adr2>,<adr3>   COPY MEMORY
 O,<adr1>,<adr2>,<file>   OUTPUT OBJECT TO FILE
 P,<adr1>,<adr2>,<byte>   PROGRAM EPROM
 S{,<step>}               SET STEP
 T{,<eprom>}              SPECIFY EPROM TYPE
 V,<adr1>,<adr2>,<byte>   VERIFY EPROM WITH MEMORY
*_
```

BURN302 blanks the internal utility buffer initially and also tests the 302 board's jumpers to decide what type of EPROM is to be burned. A detailed description of each of the BURN302 commands is listed below:

0,<file>{,<adr>}

LOAD BINARY FILE. A PDOS file is loaded into the utility buffer beginning at address <adr>.

LOAD BINARY FILE

(13.5 BURN302 continued)

1,<file>{,<adr>}

LOAD OBJECT FILE. A TI9900 tag object file is loaded into the utility buffer. The load address is dictated by the object itself or by <adr> for relocatable object. Relocatable object defaults to the location following the last store.

LOAD OBJECT FILE

2,<adr1>,<adr2>,<byte>

LOAD EPROM DATA. The EPROM inserted in the 302's configuration module is read into the utility buffer beginning at buffer address <adr1>. <byte> indicates how the data is stored. '0' or 'B' indicates a contiguous store of both bytes. '1' or 'L' reads the data into the left bytes only of the utility buffer while '2' or 'R' reads the data into the right bytes.

LOAD EPROM DATA

A

VERIFY BLANK EPROM. The EPROM inserted in the 302' configuration module is read and compared to >FF's. Any non-blank locations in the EPROM are listed to the console. If the entire EPROM is blank, then the '*' prompt is the only output.

VERIFY BLANK EPROM

B{,<adr>}

SET/SHOW BUFFER BASE. The utility buffer is a window into the 64k address space. The base is specified by <adr> and the maximum is dictated by the task memory limits. The default buffer base is address >0000. If no <adr> is specified, the address of the current base is displayed along with the current buffer logical limits.

SET/SHOW BUFFER BASE

C,<adr1>,<adr2>{,<adr3>}

COMPUTE CHECKSUM. If only two address are entered, the utility buffer memory from <adr1> up to <adr2> is summed and the result is displayed to the console. If all three address are specified, then location <adr3> is zeroed, buffer memory from <adr1> up to <adr2> is summed, and the negative of the result is stored in location <adr3>. Thus EPROM checksums can be computed, stored, and checked before burning.

```
COMPUTE CHECKSUM
*C,F800,FFFF
CHECKSUM=74E4
*C,F800,FFFF,F812
*C,F800,FFFF
CHECKSUM=0000
```

(13.5 BURN302 continued)

E

EXIT TO PDOS. The 'E' command exits to the PDOS monitor.

EXIT TO PDOS

I{,<adr>}

SET EPROM INDEX. All references to the EPROM address is based on an index into the EPROM. This is initially zero, so all burning, verifying, and reading is done from the beginning of the EPROM. To alter the index, this command sets a hex index which is an absolute byte count from the beginning of the ROM.

SET EPROM INDEX

```
*I
INDEX=0000
*I=100
*_
```

M
M,<adr1>
M,<adr1>,<adr2>
M,<adr1>,<adr2>,<adr3>

MODIFY BUFFER MEMORY. If either no parameters or one address is specified, BURN302 enters a utility buffer inspect and change mode. A location is opened by entering an address followed by a <CR>. New data could then be entered and/or the location closed by a <CR> again. A minus character acts as a <CR> except that the address is decremented by two. A space increments the address by two. A ^C exits to the utility command mode. If two parameters are specified, the utility buffer memory from <adr1> through <adr2> is displayed to the user console. If all three parameters are specified, the contents is the utility buffer memory from <adr1> through <adr2> is copied into a block beginning at address <adr3>. This the copy command.

MODIFY BUFFER MEMORY
Inspect and change

Buffer dump

Buffer copy

O,<adr1>,<adr2>,<file>

OUTPUT OBJECT TO FILE. The utility buffer contents from location <adr1> up to <adr2> is output in object code tag format to the file named <file>. The address tag for locating the absolute data is computed relative to the buffer's logical base. This command is useful for reading data from EPROMs that have been generated elsewhere.

OUTPUT OBJECT TO FILE

P,<adr1>,<adr2>,<byte>

PROGRAM EPROM. An EPROM is programmed with data from the utility buffer beginning at address <adr1> through and including <adr2>. <Byte> specifies how the data is retrieved from memory. '0' or 'B' specifies a true memory image (both bytes) while '1' or 'L' specifies the left bytes only and '2' or 'R' for the right bytes. Special counters are maintained such that the next programming parameters can be automatically set up and displayed by entering a ^P. Also, entering a ^A will repeat the previous executed command line.

PROGRAM EPROM

(13.5 BURN302 continued)

S{,<step>}

SET DEFAULT STEP SIZE. The default increment from one burn to the next is specified by <step>. For burning sequential pairs of 2716's for an application, the step size would be >0FFF.

SET DEFAULT STEP SIZE

T{,<eprom>}

SPECIFY EPROM TYPE. The type of EPROM to be programmed by the TM990/302 is specified by <eprom>. Valid types are 2516, 2708, 2516, 2716, and 2532. A carriage alone displays the current EPROM type.

SPECIFY EPROM TYPE

V,<adr1>,<adr2>,<byte>

VERIFY EPROM WITH MEMORY. An EPROM can be verified with the utility buffer by specifying the memory addresses <adr1> and <adr2> along with the type of verify <byte>. '0' or 'B' specifies both bytes while '1' or 'L' specifies the left bytes only and '2' or 'R' for the right bytes. The last programming parameters can automatically be retrieved by entering a ^V.

VERIFY EPROM WITH MEMORY

Examples:

```
.BURN302
BURN302 R2.4
*NOTE: ALL NUMBERS ARE HEX.
*T,2708
TYPE WAS 2708
*B,2000
*B,F000
HIGHEST PC=0000
BUFFER LIMITS ARE 2000 TO 9074
*1,PRGM1
 IDT='PRGMR2.0'
ENTRY ADDRESS=0000
*S,3FF
*M
2000: FFFC 7FFC
2000: 7FFC ^C
*P,2000,23FF,L
..........+
..........+
..........+
..........+
..........+
...^C
*V,2000,23FF,1
*_
```

13.6 BURNP

Name: BURNP
Function: Burn EPROMs over RS232 to ProLog
Format: BURNP

Systems: All
Language: Assembly
Memory: 4K bytes + buffer

Restrictions: Output format conforms to ProLog M900 burner

Description:

The BURNP utility loads TI9900 tag object from a PDOS file and formats it for the ProLog EPROM programmer. The base address of the memory buffer is specified by 'BEGINNING ADDRESS TO LOAD='. Several files can be loaded at once. The step size is used to generate default programming parameters. Port #2 is used to send the output to the PROLOG programmer. A ^C will stop the send process.

ProLog M900 PROM PROGRAMMER

Examples:

```
.BURNP
PROLOG TI9900 OBJECT BURN R2.1
FILE NAME=PRGM1
BEGINNING ADDRESS TO LOAD=___ <cr>
PLEASE WAIT.....
0000
FILE NAME=PRGM2
PLEASE WAIT.....
0070 0020 0083 0088 0089 008E 008F
FILE NAME=<cr>
ENTER STEP SIZE 03F
BEGINNING ADDRESS=0000
     LAST ADDRESS=009E
BEGINNING ADDRESS TO BURN=0000
     LAST ADDRESS TO BURN=003F
0 = BOTH BYTES
1 = LEFT BYTES
2 = RIGHT BYTES
3 = RIGHT NIBBLE
OPTION=1
000
01F
P
C000C0002F04CC0406152F00040A49202F00C02F2F2F00C02F2F2F00C02F2FC1
LAST BEGINNING ADDRESS=0000
     LAST LAST ADDRESS=003F
          LAST OPTION=0001
BEGINNING ADDRESS=0000
     LAST ADDRESS=009E
BEGINNING ADDRESS TO BURN=_
```

13.7 BXREF

Name: BXREF
Function: Cross references BASIC programs
Format: BXREF

Systems: All
Language: BASIC
Memory: 8k bytes minimum

Restrictions: Programs must be typed 'EX'

Description:

BXREF provides a cross reference for PDOS BASIC programs. The output includes a transfer cross reference as well as a variable cross reference.

Examples:

.SF HPLOT

```
100  W=50
110  INPUT "NUMBER OF SLICES=";M
120  YO=INT[36/M+0.5]
130  XO=INT[3*YO/5+0.5]
140  FOR X1=0 TO W+M*XO
150   J=0
160   FOR N=0 TO M
170    IF N<INT[(X1-W)/XO+0.5]: GOTO 280
180    IF N>INT[X1/XO+0.5]: GOTO 290
190    Y1=INT[N*YO+0.5]
200    IF Y1<=J: GOTO 230
210    PRINT  TAB Y1-1;".";
220    J=Y1
230    X=X1-N*XO: Y=N*SQR[XO^2+YO^2]: Z=50*FNFUN[X,Y]
240    Y1=INT[Z+N*YO+0.5]
250    IF Y1<=J: GOTO 280
260    PRINT  TAB Y1-1;"*";
270    J=Y1
280   NEXT N
290   PRINT
300  NEXT X1
1000  DEFN FNFUN[X,Y]
1010  FNFUN=EXP[-0.015*((X-25)*(X-25)+(Y-25)*(Y-25))]
1020  FNEND
```

(13.7 BXREF continued)

```
.BXREF

* BXREF *    11/25/80

1) CROSS REFERENCE FOR TRANSFER OF CONTROL
2) CROSS REFERENCE FOR VARIABLES

   SELECT OPTION: 1

TRANSFER XREF   11/25/80
ENTER FILENAME: HPLOT
ENTER UNIT FOR XREF (0-3): 3

TRANSFERS IN FILE HPLOT

  230:          200
  280:          170      250
  290:          180

* BXREF *    11/25/80

1) CROSS REFERENCE FOR TRANSFER OF CONTROL
2) CROSS REFERENCE FOR VARIABLES

   SELECT OPTION: 2

TRANSFER XREF   11/25/80
ENTER FILENAME: HPLOT
ENTER UNIT FOR XREF (0-3): 3

VARIABLES IN FILE HPLOT

FNFUN           230     1000    1010*
  J             150*    200     220*    250     270*
  M             110     120     140     160
  N             160*    170     180     190     230
                230     240     280
  TAB           210     260
  W             100*    140     170
  X             230*    230     1000    1010    1010
  X0            130*    140     170     180     230
                230
  X1            140*    170     180     230     300
  Y             230*    230     1000    1010    1010
  Y0            120*    130     190     230     240
  Y1            190*    200     210     220     240*
                250     260     270
  Z             230*    240
```

13.8 CHATLE

Name: CHATLE
Function: Changes attributes and levels of selected files.
Format: CHATLE
CHATLE ∂:∂;∂/‹disk #›,{‹attribute›},{‹level #›}

Systems: All
Language: Assembly
Memory: 6K bytes

Restrictions: Cannot use level 255.

Description:

The CHATLE utility changes the attributes and/or the directory level of a selected group of files to the specified value. If neither is specified, then the allotted and used disk space is printed. The mnemonic is CHange ATtributes and LEvels, for CHATLE. The file descriptor string is the same as that used in TRANS and LDIR. An '∂' indicates a wild card of all possible selections and a '*' is a single character wild card.

The attribute parameter must either be one of the PDOS defined file types (AC, EX, BX, OB, SY, BN, UD, or TX), a protection flag (* or **), a pound sign #, or an at-sign ∂. If a PDOS attribute is specified (file type and/or protection flag), then all files matching the selection list are given those attributes. If a '#' is specified, the files' contiguous flags are cleared. If an '∂' is specified, then the protection flags are cleared.

The level parameter, if present, must be a number from 0 to 254. All files matching the selection list are assigned to the specified level. The parameters can either be passed to CHATLE in the command line or by prompts from the utility.

(13.8 CHATLE continued)

Examples:

```
.LDIR/1 T@:@                                                    See the files.
DISK NAME=NEWER NAME/0                                 FILES=35/128
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 1   TEMP            *       2/49     14:49 09/21/82  19:10 09/24/82
 1   TEMP1                   1/9      08:57 09/23/82  18:13 09/24/82
 1   TRANS          SY       15/15    21:51 09/16/82  21:51 09/16/82
.CHATLE T@:@;@,**,55                                            Set to **, level 55.
.LS 55                                                          See them on 55.
DISK=NEWER NAME/0                                      FILES=35/128
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 55  TEMP            **      2/49     14:49 09/21/82  19:10 09/24/82
 55  TEMP1           **      1/9      08:57 09/23/82  18:13 09/24/82
 55  TRANS          SY **    15/15    21:51 09/16/82  21:51 09/16/82
.CHATLE T@:@;@                                                  List level 55 size
USED=18/73
.CHATLE                                                         Invoke menu.
CHANGE DIRECTORY ATTRIBUTES/LEVELS R2.4
MASK={<name>}{:<ext>}{;<level>}{/<disk #>}{/F<date>}{/T<date>}  Hit <cr>, get format.
ATTRIBUTES={<type>}{<flags>}{#=clear 'C'}{@=clear **}
LEVEL={<0-254>}                                                 Note: no level 255.
CHANGE DIRECTORY ATTRIBUTES/LEVELS R2.4
MASK=TE@:@;@                                                    Just TEMP files.
ATTRIBUTES=TX@                                                  To TX, w/o protection.
LEVEL=<cr>
.LS 55                                                          See the difference.
DISK=NEWER NAME/0                                      FILES=35/128
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 55  TEMP           TX        2/49     14:49 09/21/82  19:10 09/24/82
 55  TEMP1          TX        1/9      08:57 09/23/82  18:13 09/24/82
 55  TRANS          SY **     15/15    21:51 09/16/82  21:51 09/16/82
.-
```

13.9 COMP

Name: COMP
Function: Compares pairs of ASCII files
Format: COMP

Systems: All
Language: BASIC
Memory: 8K bytes minimum

Restrictions: none

Description:

The COMP utility compares two ASCII files and prints any differences. This is extremely useful in documenting updates to source programs. The utility reads as many lines as possible from each file and begins comparing. If a difference is found, the program then searches for at least three consecutive lines that are the same in each file. This would then 'sync' the compare again.

Only 38 characters of each line are printed. TAB characters are reduced to a single space. Any lines deleted or inserted in a file will be lined up by a row of dots with the other file.

Three blank lines will follow after each block of differences. The end of the program prints the number of differences encountered.

If the program cannot get in 'sync', the message 'TOO MANY DIFFERENCES FOR AVAILABLE MEMORY' is printed.

Examples:

```
.COMP
*** COMPARE ***
FILE 1: TEMPA
FILE 2: TEMPB/1
OUTPUT FILE=<cr>


FILE TEMPA                          FILE TEMPB/1
==================================  ==================================

 10  REM EXAMPLE OF COMPARE PROGRAM     10  REM EXAMPLE OF COMPARE PROGRAM
 20  FOR I=1 TO 100                     20  FOR I=1 TO 200
 30   IF I=2: GOTO 50                   ..................................
 40   IF I=18: GOTO 50                  ..................................
 50  NEXT I                             50  NEXT I

 1 DIFFERENCES FOUND

STOP AT 4030
```

13.10 DDMAP

Name: DDMAP
Function: Disk diagnostic: reads files by links
Format: DDMAP

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: none

Description:

The DDMAP utility provides a comprehensive PDOS disk map for disk diagnostics and file repair. File links are displayed as well as damaged sectors and spoiled bit maps.

The program displays the disk name, the number of directory entries, date initialized, the number of PDOS sectors and the disk density. A table is created from the disk sector bit map which will be compared with the sectors allocated as indicated by the file links.

Next, the files from the disk directory are processed. The file directory entry is listed followed by the sector maps. The start sector of a contiguous block of sectors is listed followed by the end sector number. File links are followed until a null link is encountered. As each sector is read, a new bit map is created as well as the old map checked. If the sector has already been allocated, then the sector is listed in brackets indicating a spoiled file (more than one file claiming that sector.) If the sector is not allocated in the old map, then it is indicated in a similar fashion. Any file I/O errors are also listed.

In this manner, the whole disk is processed and checked for possible file contaminations. The information is useful in physically locating where files begin and end according to sector numbers.

```
.DDMAP/4
DISK DIAGNOSTIC MAPPER R2.4
DISK #=5
OUTPUT=<cr>
*DISK DIAGNOSTIC MAP

DISK NAME=WIND #1
FILES=64/256
CREATED=10:01  09/15/82
PDOS SECTORS=8144
DISK DENSITY=0

 0   ASM          SY       48/48   08:15 07/...
     3205-3252

 60  BOOT         OB       22/22   15:02 08/...
     3923-3944

 9   DO           AC        3/3    08:02 07/...
     3555-3556   3576-3576

 10  DOCUDO                66/75   10:38 07/...
     3559-3575   4767-4770   4913-4925   4938...

 10  LINKMAP      TX       20/20   08:23 07/...
     3272-3272   3589-3590   3695-3700   3702...

 1   LIST         OB       17/1572 15:47 07/...
     37-1436     2658-2829

 1   LIST1        OB       19/1221 15:56 07/...
     1437-1846   12252-[12252]
     ****** PDOS READ ERROR 233 AT SECTOR 12252

 10  MODULE       OB      119/121  08:23 07/...
     3271-3271   3579-3588   3591-3694   3701...

 9   PAUSE        OB        1/1    08:12 07/...
     3578-3578

 9   PAUSE:SR     TX        1/1    08:02 07/...
     3557-3557

.
```

13.11 DDUMP

Name: DDUMP
Function: Disk sector dump and alter
Format: DDUMP

Systems: All
Language: Assembly
Memory: 8K bytes

Restrictions: Can re-write sectors external to PDOS

Description:

The DDUMP utility dumps in hex and ASCII format sectors from a disk. A sector alter mode allows reading and writing individual sectors. The program prompts for disk unit, start sector, and end sector number. The sector number is displayed in both hex and (decimal) representation as well as the unit number at the beginning of each sector display.

The next sector can immediately be selected with ^N, thus aborting the display of the whole sector. A ^D will revert back to the start sector prompt. The list can be temporarily stopped by striking the space bar. An <esc> will exit DDUMP.

If the letter A is entered for the starting sector prompt, the sector alter mode is initiated. Alter mode asks for the sector number to alter. A control C will return to the main DDUMP program, asking for start sector. After entering the alter sector number, the sector is read into the alter buffer, the buffer is displayed, and the cursor is placed over the first byte of the sector. The same characters used to move the cursor in JED also work in alter mode. Entering hex byte data changes the buffer data. To write the sector to the disk, enter control W. DDUMP then asks for the sector number to write the alter buffer to. A carriage return here writes the buffer to the same sector number that was read in last. The answer to the verify prompt is 'V'. The utility then prompts again for a sector number to read.

Alter mode
^C to exit
^W to write out sector

(13.11 DDUMP continued)

Examples:

```
.DDUMP
DISK DUMP R2.1
DISK=0
START SECTOR=0
END SECTOR=1
 SECTOR/DISK=>0000 (0)/0
 000-00F   5041 554C 2023 314D 4400 0000 0000 4000 PAUL #1MD.......
 010-01F   0007 0011 0000 0000 0040 02C7 A55A FFFF .........@.G%Z..
 020-02F   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 030-03F   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 040-04F   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 050-05F   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 060-06F   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 070-07F   FFFF FFFF C000 0000 007F FFFF FFFF FFFF ....@...........
 080-08F   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 090-09F   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 0A0-0AF   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 0B0-0BF   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 0C0-0CF   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 0D0-0DF   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 0E0-0EF   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 0F0-0FF   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
 SECTOR/DISK=>0001 (1)/0
 000-00F   4845 4144 4552 0000 0000 0001 0000 0260 HEADER.........`
 010-01F   0000 000B 000B 00FA 062D AE50 062E AE50 .......z.-.P...P
 020-02F   4A45 4442 0000 0000 0000 0001 0000 0171 JEDB...........q
 030-03F   0000 0029 0029 00FC 0513 A750 05A7 BCD0 ...).).¦..'P.'<P
 040-04F   414C 4152 4D00 0000 0000 0001 0400 0294 ALARM...........
 050-05F   0000 0000 0000 00DE 06AD AF50 0181 AFD0 .......^.-/P../P
 060-06F   444F 4330 3200 0000 0000 0001 0000 00CB DOC02..........K
 070-07F   0000 005C 005C 0005 050F A750 05A0 B1D0 ...\.\....'P. 1P
 080-08F   444F 4330 3300 0000 0000 0001 0000 00F3 DOC03..........s
 090-09F   0000 0049 0049 000B 0510 A750 0A80 A6D0 ...I.I.[..'P..&P
 0A0-0AF   444F 4330 3400 0000 0000 0001 0000 0137 DOC04..........7
 0B0-0BF   0000 0049 0049 0092 0512 A750 0A90 A6D0 ...I.I....'P..&P
 0C0-0CF   444F 4330 3500 0000 0000 0001 0000 0172 DOC05..........r
 0D0-0DF   0000 004C 004C 00ED 0514 A750 0527 B750 ...L.L.m..'P.'7P
 0E0-0EF   2454 5441 5300 0000 0000 0000 0000 0291 $TTAS...........
 0F0-0FF   0000 0000 0000 00FC 0729 AED0 0729 AED0 .......¦.).P.).P
START SECTOR=_
```

13.12 DISCAT

Name: DISCAT
Function: Catalogs combined directories of multiple disks.
Format: DISCAT

Systems: All
Language: Assembly
Memory: 16K bytes minimum

Restrictions: Number of files cataloged limited by memory space.

Description:

The DISCAT utility catalogs disk directory information from multiple disks. Output of DISCAT can be by files (TYPE=1), by disks (TYPE=2), or both (TYPE=3).

Disk catalog information includes:

DISK	Number given to disk for directory reference
NAME	Disk header name
S/D	Disk sides/disk density
SZE	Number of PDOS sectors on disk
NDE	Number of directory entries available
USD	Number of sectors used
ALL	Number of sectors allocated to file storage
FRE	Number of sectors free on disk
BAD	SZE - (FRE + ALL)
DATE CREATED	Date disk initialized

Disk directory information includes:

DISK	Disk on which the file is located
NAME:EXT	File name and extension
LEV	Directory level
TYPE	PDOS file type
FLG	Contiguous or protection flags
SIZE	Sectors used vs. allocated
DATE CREATED	Date file defined
LAST UPDATE	Date file last written to

DISCAT first prompts for DISK and then asks if disk is ready. A 'Y' response catalogs the disk by reading all the directory entries on the disk and merging them into a linked list. A <CR> reverts back to the 'DISK' prompt. A 'N' response moves to the 'OUTPUT FILE=' prompt. If no output file is entered (just <CR>), the directory information is listed on the user console. The final prompt asks for catalog type. Type 1 produces a catalog by files, which is a single, alphabetized list, merging all directory entries read. Type 2 produces a catalog by disks, where an alphabetized list is output for each disk, along with sector information. Type 3 outputs both types of lists.

(13.12 DISCAT continued)

Examples:

```
.DISCAT
DISK CATALOG R2.1
DISK #=0
NEXT DISK READY?Y........DONE
NEXT DISK READY?Y........DONE
NEXT DISK READY?<cr>
DISK #=1
NEXT DISK READY?Y........DONE
NEXT DISK READY?N
OUTPUT FILE=<cr>
OUTPUT TYPE=3

***  DISK CATALOG  ***                                          DATE=03/13/81
DISK NAME               S/D  SZE   NDE   USD   ALL   FRE   BAD   DATE CREATED

  1  PAUL #3MD          S/D  712    64   607   649    54     0  13:10 03/03/81
  2  PAUL #4MD          S/D  712    64   633   662    41     0  13:22 03/05/81
  3  PAUL #5MD          S/D  712    64   547   572   131     0  15:07 03/05/81

***  DIRECTORY CATALOG  ***                                     DATE=03/13/81
DISK NAME:EXT           LEV  TYPE FLG  SIZE      DATE CREATED    LAST UPDATE

  1  $TTA                 0                 1/1   11:12 03/07/81  11:12 03/07/81
  1  DOC00                1   TX  *      73/95   12:36 01/05/81  13:18 03/12/81
  1  DOC01                1   TX  *      43/43   15:02 03/03/81  13:21 03/12/81
  1  DOC02                1   TX  *      88/95   12:37 01/05/81  10:02 03/13/81
  1  DOC03                1   TX  *      94/94   12:37 01/05/81  13:26 03/12/81
  1  DOC04                1   TX  *      91/91   12:38 01/05/81  10:04 03/13/81
  1  DOC05                1   TX  *      91/94   12:38 01/05/81  10:06 03/13/81
  2  DOC06                1               77/77   13:12 01/05/81  13:39 03/12/81
  2  DOC07                1               87/91   13:12 01/05/81  13:40 03/12/81
  2  DOC08                1               60/60   13:13 01/05/81  13:43 03/12/81
  2  DOC09                1               58/58   13:14 01/05/81  13:45 03/12/81
  2  DOC10                1               88/94   13:14 01/05/81  10:08 03/13/81
  2  DOC11                1               83/83   13:15 01/05/81  13:51 03/12/81
  2  DOC12                1               74/74   13:15 01/05/81  10:09 03/13/81
  2  DOC13                1               81/83   13:16 01/05/81  13:56 03/12/81
  3  DOC14                1               65/67   13:23 01/05/81  13:57 03/12/81
  3  DOC15                1               54/54   13:24 01/05/81  13:58 03/12/81
  3  DOC16                1               93/93   13:24 01/05/81  13:59 03/12/81
  3  DOC17                1               91/92   13:25 01/05/81  11:39 03/13/81
  3  DOC18                1               34/34   10:46 03/09/81  10:21 03/13/81
  1  DOCHD                1   TX  *      15/15   12:39 01/05/81  08:39 03/06/81
  1  DOCTC                1   TX  *      65/75   12:39 01/05/81  12:25 03/12/81
  1  JEDY                 0   SY  **     25/25   10:42 03/07/81  10:43 03/07/81
  2  JEDY                 1   SY         25/42   13:11 01/05/81  13:24 03/05/81
  3  JEDY                 1   SY         25/25   11:37 03/08/81  11:38 03/08/81
  1  PRINTS               2   EX         21/21   12:40 01/05/81  15:49 03/05/81
  3  temp                 1                 1/1   10:14 03/13/81  11:32 03/13/81
```

(13.12 DISCAT continued)

```
***  DISK CATALOG  ***                                          DATE=03/13/81
DISK NAME              S/D  SZE   NDE   USD   ALL   FRE   BAD   DATE CREATED

  1  PAUL #3MD         S/D  712    64   607   649    54     0  13:10 03/03/81

***  DIRECTORY CATALOG  ***                                     DATE=03/13/81
DISK NAME:EXT          LEV  TYPE FLG  SIZE      DATE CREATED    LAST UPDATE

  1  $TTA                0             1/1     11:12 03/07/81  11:12 03/07/81
  1  DOC00               1   TX  *    73/95     12:36 01/05/81  13:18 03/12/81
  1  DOC01               1   TX  *    43/43     15:02 03/03/81  13:21 03/12/81
  1  DOC02               1   TX  *    88/95     12:37 01/05/81  10:02 03/13/81
  1  DOC03               1   TX  *    94/94     12:37 01/05/81  13:26 03/12/81
  1  DOC04               1   TX  *    91/91     12:38 01/05/81  10:04 03/13/81
  1  DOC05               1   TX  *    91/94     12:38 01/05/81  10:06 03/13/81
  1  DOCHD               1   TX  *    15/15     12:39 01/05/81  08:39 03/06/81
  1  DOCTC               1   TX  *    65/75     12:39 01/05/81  12:25 03/12/81
  1  JEDY                0   SY  **   25/25     10:42 03/07/81  10:43 03/07/81
  1  PRINTS              2   EX       21/21     12:40 01/05/81  15:49 03/05/81

***  DISK CATALOG  ***                                          DATE=03/13/81
DISK NAME              S/D  SZE   NDE   USD   ALL   FRE   BAD   DATE CREATED

  2  PAUL #4MD         S/D  712    64   633   662    41     0  13:22 03/05/81

***  DIRECTORY CATALOG  ***                                     DATE=03/13/81
DISK NAME:EXT          LEV  TYPE FLG  SIZE      DATE CREATED    LAST UPDATE

  2  DOC06               1             77/77     13:12 01/05/81  13:39 03/12/81
  2  DOC07               1             87/91     13:12 01/05/81  13:40 03/12/81
  2  DOC08               1             60/60     13:13 01/05/81  13:43 03/12/81
  2  DOC09               1             58/58     13:14 01/05/81  13:45 03/12/81
  2  DOC10               1             88/94     13:14 01/05/81  10:08 03/13/81
  2  DOC11               1             83/83     13:15 01/05/81  13:51 03/12/81
  2  DOC12               1             74/74     13:15 01/05/81  10:09 03/13/81
  2  DOC13               1             81/83     13:16 01/05/81  13:56 03/12/81
  2  JEDY                1   SY        25/42     13:11 01/05/81  13:24 03/05/81

***  DISK CATALOG  ***                                          DATE=03/13/81
DISK NAME              S/D  SZE   NDE   USD   ALL   FRE   BAD   DATE CREATED

  3  PAUL #5MD         S/D  712    64   547   572   131     0  15:07 03/05/81

***  DIRECTORY CATALOG  ***                                     DATE=03/13/81
DISK NAME:EXT          LEV  TYPE FLG  SIZE      DATE CREATED    LAST UPDATE

  3  DOC14               1             65/67     13:23 01/05/81  13:57 03/12/81
  3  DOC15               1             54/54     13:24 01/05/81  13:58 03/12/81
  3  DOC16               1             93/93     13:24 01/05/81  13:59 03/12/81
  3  DOC17               1             91/92     13:25 01/05/81  11:39 03/13/81
....
```

13.13 DISCVER

Name: DISCVER
Function: Disk diagnostic: non-destructive sector verifier.
Format: DISCVER

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: none

Description:

The DISCVER utility is non-destructive disk diagnostic that reads block of sectors and keeps track of all errors encountered. The utility prompts for the disk number, beginning sector number, and the ending sector number. All sectors within the specified range are then read, once each pass. The order in which they are read alternates between an increasing sequential order and a random order. The random passes help check stepping and settling times for floppies and Winchesters. The program can be aborted at any time by entering <ESC>. An intermediate list of sector errors is printed when a ^C is entered.

Random and sequential order

<ESC> to abort
^C for intermediate error list

The utility uses the read sector primitive of PDOS, and all errors are reported as they occur, along with the number of the offending sector. Internally, a bit record is kept of each sector's performance: one bit is set on if a sector returns an error and another bit is set if a sector is read without error.

At the end of each pass, the pass number and number of total errors are output. A list of the sectors which have ever returned errors follows, along with the count in angle brackets <>. DISCVER then outputs a message indicating the sectors which have never been read without an error being returned, along with the <count>. If all sectors have been read at least once without error, that message is output. This is followed by a summary of the error numbers reported by the DSRs and the number of times each error number was returned.

The DISCVER utility works on any PDOS disk device, and it runs indefinitely, until interrupted by the operator (^C).

(13.13 DISCVER continued)

Examples:

```
.DISCVER
DISK VERIFIER R2.4
DISK #=0
BEGINNING SECTOR #=0
ENDING SECTOR #=1975
END PASS # 1     TOTAL ERRORS = 0
Sectors which have had errors : <0>
Sectors which have NEVER been read :    <0>


END PASS # 2     TOTAL ERRORS = 0
Sectors which have had errors : <0>
All sectors have been read at least once.

...
```

13.14 DNAME

Name: DNAME
Function: Renames PDOS disks
Format: DNAME
DNAME <disk #>,<new name>

Systems: All
Language: Assembly
Memory: 1K bytes

Restrictions: none

Description:

The DNAME utility renames PDOS disks by altering the header sector (sector 0) of the disk. DNAME prompts for the disk number, reads the header sector of the desired disk, and reports the old name. DNAME then prompts for the new name and writes out the header sector again with the new name.

OLD NAME=
NEW NAME=

An alternate way to rename a disk with this utility is to follow the DNAME call with two parameters in the command line. The first is the disk number and the second is the new disk name desired. DNAME outputs the old name and renames the disk with no other action. Note that the alternate method does not allow the use of commas in the new name, since the command line interpreter uses the comma to delimit parameters (blanks are OK). The new name would consist of only those characters preceding the comma.

DNAME 0,NEW NAME

Examples:

```
.LS
DISK=PDOS DISK OLD/0                                    FILES=27/128
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 1   JEDY          SY        26/26     21:51 09/16/82  21:52 09/16/82
 1   TEMP                     2/49      14:49 09/21/82  20:50 09/22/82
 1   TRANS         SY        15/15     21:51 09/16/82  21:51 09/16/82
.DNAME
DISK NAME R2.4
DISK #=0
OLD NAME=PDOS DISK OLD
NEW NAME=PDOS DISK NEW
.LS
DISK=PDOS DISK NEW/0                                    FILES=27/128
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 1   JEDY          SY        26/26     21:51 09/16/82  21:52 09/16/82
 1   TEMP                     2/49      14:49 09/21/82  20:50 09/22/82
 1   TRANS         SY        15/15     21:51 09/16/82  21:51 09/16/82
```

(13.14 DNAME continued)

```
.DNAME 0,NEWER NAME
HAS PDOS DISK NEW
.LS
DISK=NEWER NAME/0                                    FILES=27/128
LEV  NAME:EXT      TYPE     SIZE     DATE CREATED    LAST UPDATE

 1   JEDY          SY      26/26    21:51 09/16/82  21:52 09/16/82
 1   TEMP                   2/49    14:49 09/21/82  20:50 09/22/82
 1   TRANS         SY       15/15   21:51 09/16/82  21:51 09/16/82
.
```

13.15 DRGN

Name: DRGN
Function: Generates PDOS I/O drivers from TI object code.
Format: DRGN
DRGN <infile>,<binfile>

Systems: All
Language: Assembly
Memory: 2K bytes

Restrictions: none

Description:

The DRGN utility generates a PDOS I/O driver from TI 9900 object code. The program resolves the object code into address-independent binary code, checking for illegal tags and correct driver size. The use of this utility is described in detail in Chapter 7, which contains a complete discussion of I/O drivers in general.

Examples: See Chapter 7.

13.16 EDIT

Name: EDIT
Function: Character editor with a macro.
Format: EDIT
EDIT <filename>

Systems: All
Language: Assembly
Memory: 4K bytes + buffer

Restrictions: File size is limited by memory.

Description:

The EDIT utility is a character oriented file editor with macro capability. This is one of the three editors supplied with PDOS and is useful for editing on terminals with no cursor addressing or with incompatible function codes. This rather simple editor comes in handy when you want the alter a file under the control of a chain file (type AC). A complete description of the EDIT utility is found in Chapter 11. The following is a summary of the EDIT commands:

A	Append next page
B	Move to beginning of text buffer
C<s1>$<s2>	Change s1 to s2 once
#D	Character delete
GI<s>	Get file <s> for input
GI	Close input file
GO<s>	Get file <s> for output
GO	Close output file
H	Return to PDOS
#I	Insert byte
I<s>	Insert string <s>
#J	Move lines (block)
#K	Kill (delete) lines
#L	Line move {P}
#M	Character move {P}
N	New buffer
P	Punch buffer
#P	Punch # lines
PH	Punch without FF
#PH	Punch # lines without FF
S<s>	Search for string <s> once
T	Type entire buffer
#T	Type # lines {P}
#X	Execute MACRO
XM<s>	Define MACRO
XM	Delete MACRO
Y	Yank new page (to FF)
Z	Move to end of text buffer
.	HERE pointer
:	# of lines
=	# of characters
+	# of characters left

Examples: See Chapter 11.

13.17 ER3314

Name: ER3314
Function: Diagnostic: Test for ER3314 SASI Winchester adapter.
Format: ER3314

Systems: 101,102
Language: Assembly
Memory: 8K bytes

Restrictions: May destroy data on attached disks.

Description:

The ER3314 utility is a diagnostic program for Eyring's ER3314 SASI host adapter board for the TM990 bus. This card interfaces a Shugart Associates System Interface (SASI) compatible Winchester controller with either the 101 of 102 PDOS systems. The ER3314 utility allows users of the board to perform a random read/write sector, a single sector inspect and change, a controller reset, a read status of drive, or send a command to the controller. When the utility is run, the following menu is printed:

```
ER3314 Diagnostics, R2.4
  1. Random sector R/W.
  2. Single sector inspect & change.
  3. Reset DTC.
  4. Read DTC status.
  5. Send DTC command.
  6. Dual GPIB check.
Input Selection:
```

The first selection prompts for a logical unit number (0-3), a beginning sector number, an ending sector number, and the method of data transfer (0,1,2,3, or 7). The logical unit number is the controller's logical unit and NOT the PDOS disk number. The sector numbers refer to the 'block' number of the controller, NOT the logical sector number of PDOS. The transfer type input is a bit coded variable: the LSB set to a one performs a DMA read, a polled I/O read otherwise; the next LSB set to a one performs a DMA write, a polled I/O write otherwise; and the next bit, the 4 bit, performs a wait for interrupt to signal completion. The last bit can only be used if the interrupt on the ER3314 is jumpered to level 2 and both read and write are selected as DMA operations, (select a 7 for this). This selection writes random data to random sectors within the specified range to the logical unit selected. An escape (<ESC>) aborts a 'locked up' state and a control C (^C) halts the R/W and returns to the main menu. Entering a 'P' toggles the sector print on and off, with only errors being printed.

1=Random read/write

NOTE: The random R/W destroys data on the selected drive. Beware!

Destroys data!!

(13.17 ER3314 continued)

Selection 2 performs an interactive sector read, alter, and write function. The logical unit, the sector number and the transfer type are requested. The sector is then read into an edit buffer,
the data is displayed, and the cursor is placed over the first byte of the sector. This alter mode is much like the alter mode of the DDUMP utility.

2=Sector read, alter, write

Selection 3 performs a hardware reset of the SASI controller attached to the ER3314.

3=RESET

Selection 4 issues a 'READ SENSE' for the logical unit requested and reports the 6 bytes returned from the controller.

4=Read sense

Selection 5 issues the 6 or 10 byte command input by the user. It can also read or write any number of bytes after the specified command is issued and displays them to the console.

5=Issue command

The following examples show the various inputs requested by ER3314.

Examples:

```
.ER3314
ER3314 Diagnostics, RZ.4                                  Main menu
  1. Random sector R/W.
  2. Single sector inspect & change.
  3. Reset DTC.
  4. Read DTC status.
  5. Send DTC command.
  6. Dual GPIB check.
Input Selection:1                                         Select random R/W
HIGHEST SECTOR # = 1046
LOWEST SECTOR # = 26                                      Don't read TRK00, 128 byte sectors
LUN = 2
TYPE OF TRANSFER:
      0 = POLLED R/W
      1 = DMA R/POLLED W
      2 = POLLED R/DMA W
      3 = DMA R/W
     +4 = DMA INTERRUPT
(0-7)?7                                                   Select DMA w/interrupt
```

The sector number and LUN are output along with the total number of sectors read, the total number of data discrepancies, and the total number of sectors with errors. Hit 'P' to toggle sector print and ^C to return to main menu.

(13.17 ER3314 continued)

```
>> BREAK                                                    ^C
DONE! HIT RETURN<cr>
ER3314 Diagnostics, R2.4
  1. Random sector R/W.
  2. Single sector inspect & change.
  3. Reset DTC.
  4. Read DTC status.
  5. Send DTC command.
  6. Dual GPIB check.
Input Selection:2                                          Select IAC
SECTOR INSPECT AND CHANGE
LUN = 2
TYPE OF TRANSFER:
      0 = POLLED R/W
      1 = DMA R/POLLED W
      2 = POLLED R/DMA W
      3 = DMA R/W
     +4 = DMA INTERRUPT
(0-7)?7
READ FROM SECTOR = 2000
```

The sector is read and displayed to the screen. A ^W writes the buffer to the disk, if verified, and a ^C returns to the main menu.

```
DONE! HIT RETURN<cr>
ER3314 Diagnostics, R2.4
  1. Random sector R/W.
  2. Single sector inspect & change.
  3. Reset DTC.
  4. Read DTC status.
  5. Send DTC command.
  6. Dual GPIB check.
Input Selection:3                                          Select the reset
DTC IS RESET
DONE! HIT RETURN<cr>
```

(13.17 ER3314 continued)

```
ER3314 Diagnostics, R2.4
  1. Random sector R/W.
  2. Single sector inspect & change.
  3. Reset DTC.
  4. Read DTC status.
  5. Send DTC command.
  6. Dual GPIB check.
Input Selection:4                                  Select the status read
LUN = 2
READ SENSE = 00  00  00  00                        No errors!
DONE! HIT RETURN<cr>
ER3314 Diagnostics, R2.4
  1. Random sector R/W.
  2. Single sector inspect & change.
  3. Reset DTC.
  4. Read DTC status.
  5. Send DTC command.
  6. Dual GPIB check.
Input Selection:5                                  Select the command send
# OF COMMAND BYTES = 6
# OF DATA BYTES = 0
ENTER BYTES 2 AT A TIME = C040  0000  0007         Issue a define disk parameters (SA1403D only)
DONE! HIT RETURN<cr>                               Double sided/double density 8"
ER3314 Diagnostics, R2.4
  1. Random sector R/W.
  2. Single sector inspect & change.
  3. Reset DTC.
  4. Read DTC status.
  5. Send DTC command.
  6. Dual GPIB check.
Input Selection:_
```

13.18 FDUMP

Name: FDUMP
Function: Output logical dump of PDOS files.
Format: FDUMP
FDUMP <infile>{,<outfile>}{,<increment>}

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: none

Description:

The FDUMP utility outputs a hex and ASCII dump of a PDOS file. If no output file is specified, the console terminal is used. An additional prompt is for 'INCREMENT'. This specifies how many bytes are to be listed per line. The default is 16. With 132 column output, the increment could be set to 32 to save paper.

The format of FDUMP is the file displacement, 2 hex characters per byte, followed by the ASCII characters. If an ASCII character is unprintable, then a period is used. The last line will have F's after the End-of-File is reached.

An alternate method of invoking the FDUMP utility is to pass the parameters in the command line. Follow the FDUMP call with the input file name and ,optionally, the name of the output file and the increment. The dump proceeds without any further inputs.

```
FDUMP DROPEX,LIST/5,32
FDUMP FDUMP
```

Examples:

```
.FDUMP
FILE DUMP R2.4
FILE=DROPEX
OUTPUT=
INCREMENT=16

0000-000F  4100 0042 2F5B 4300 2442 0300 4200 0042  A..B/[C.$B..B..B
0010-001F  02E0 422F DC42 02A1 4206 4142 04D1 4202  .`B/\B.!B.AB.QB.
0020-002F  8142 22AA 421B FB42 0720 422F 6A42 0207  .B"*B.{B. B/jB..
0030-003F  4230 0042 0460 4200 DC42 0A0D 4244 5242  B0.B.`B.\B..BDRB
0040-004F  4F50 4245 5842 2052 4232 2E42 3120 420A  OPBEXB RB2.B1 B.
0050-005F  0D42 5041 4255 5342 452E 422E 2E42 2E48  .BPABUSBE.B..B.H
0060-006F  4249 5442 2052 4245 5442 5552 424E 0041  BITB RBETBURBN.A
0070-007F  0047 4200 0032 0000 FFFF FFFF FFFF FFFF  .GB..2.........
.FDUMP LOAD:SR,,20

0000-0013  2A09 4C4F 4144 3A53 5209 3033 2F30 322F 3831 0D2A  *.LOAD:SR.03/02/81.*
0014-0027  2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A  **********
0028-003B  2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A 2A2A  **********
003C-004F  2A2A 2A0D 2A0D 0949 4454 2027 4C4F 4144 2032 2E31  ***.*..IDT 'LOAD 2.1
....
```

13.19 FORTH

Name: FORTH
Function: FORTH language interpreter.
Format: FORTH

Systems: All
Language: Assembly
Memory: 16K bytes minimum

Restrictions: No disk is simulated yet.

Description:

The FORTH language is a threaded, extensible interpretive language. This is the FIG-FORTH version implemented on the TMS 9900 and compatible with PDOS. The sources of the PDOS FORTH interpreter are available and the documentation is readily available from the FORTH Interest Group. The following words are in the FORTH dictionary:

!	0<	BASE	DR0	LFA	SCR
!CSP	0=	BEGIN	DR1	LIMIT	SIGN
#	0BRANCH	BL	DROP	LIST	SMUDGE
#>	1	BLANKS	DUP	LIT	SP!
#S	1+	BLK	ELSE	LITERAL	SP@
''	2	BLOCK	EMIT	LO	SPACE
(	2+	BRANCH	EMPTY-BUFFERS	LOAD	SPACES
(+LOOP)	3	BUFFER	ENCLOSE	LOOP	STATE
(.")	:	C	END	M*	SWAP
(;CODE)	;	C!	ENDIF	M/	THEN
(ABORT)	;CODE	C@	ERASE	M/MOD	TIB
(DO)	;S	CASE:	ERROR	MAX	TOGGLE
(FIND)	<	CFA	EXECUTE	MESSAGE	TRAVERSE
(LINE)	<#	CMOVE	EXIT	MIN	TRIAD
(LOOP)	<BUILDS	COLD	EXPECT	MINUS	TYPE
(NUMBER)	=	COMPILE	FENCE	MOD	U*
*	=CELLS	CONSTANT	FILL	NFA	U.
*/	>	CONTEXT	FIRST	NUMBER	U/
*/MOD	>R	COUNT	FLD	OFFSET	UNTIL
+	?	CR	FORGET	OR	UPDATE
+!	?COMP	CREATE	FORTH	OUT	USE
+-	?CSP	CSP	HERE	OVER	USER
+BUF	?ERROR	CURRENT	HEX	PAD	VARIABLE
+LOOP	?EXEC	D+	HI	PFA	VOC-LINK
,	?LOADING	D+-	HLD	PREV	VOCABULARY
-	?PAIRS	D.	HOLD	QUERY	WARNING
-->	?STACK	D.R	I	QUIT	WHILE
-DUP	?TERMINAL	DABS	ID.	R	WIDTH
-FIND	@	DECIMAL	IF	R#	WORD
-TRAILING	ABORT	DEFINITIONS	IMMEDIATE	R/W	XOR
.	ABS	DIGIT	IN	R0	[
."	AGAIN	DLITERAL	INDEX	R>	[COMPILE]
.LINE	ALLOT	DMINUS	INTERPRET	REPEAT	]
.R	AND	DO	J	ROT	
/	B/BUF	DOES>	KEY	RP!	
/MOD	B/SCR	DP	LATEST	S->D	
0	BACK	DPL	LEAVE	S0	

13.20 FREE

Name: FREE
Function: Transfer user memory back to Free Pool.
Format: FREE
FREE (-)<# (K bytes)>

Systems: All, 102 differs
Language: Assembly
Memory: 1K bytes

Restrictions: None

Description:

The FREE utility deallocates a part of the user's memory space and returns it to the pool of free memory, from which tasks are created. This utility comes in handy when you want to create tasks, but do not want to take the memory back when the tasks are killed. Since the 101 PDOS tasks try to expand their memory, if possible, whenever they return to the monitor, and since the 102 PDOS doesn't, the FREE utility acts differently in the 102 PDOS system.

In all PDOS systems other than the 102, FREE deallocates the specified amount of memory, as long as it leaves at least 1K bytes for the current task. This deallocation starts at the beginning of the task's user space. The task and its status block are moved up by the amount specified, thus reducing the size of the current task. If the parameter is not preceded with a minus sign (-), then FREE sets the corresponding bits in the memory free pool map. This memory is now available for the creation of other tasks by any task. If the parameter is preceded with a (-), then the task is moved up, the memory is available, but the bits are not set in the memory free pool map. In this way, a task may free up memory for his use only, without fear of another task stealing the memory away. In either case, this memory can be recovered by the task using the RECOVER utility.

101, STD, VG PDOS:
Free from bottom up.

No (-), add to free pool.

With (-), don't add to pool.

In the 102 PDOS system, the FREE utility rounds the amount up to a multiple of 4K and then deallocates that much beginning at the end of the task's memory. This is because the 102 PDOS doesn't try to expand its memory when it returns to the monitor. The minus (-) sign is not applicable to 102 PDOS.

102 PDOS:
Free from top down

No (-) in 102 PDOS.

(13.20 FREE continued)

Examples:

```
.LT                                                          Our task begins at >6000.
TASK  PAGE  TIME    TB    WS    PC    SR    BM    EM   CRU   PORT

*0/0   0     3    >6020 >619A >04E0 >1005 >6000 >E000 >0080 >0001
.LM
FREE=0
.FREE                                                        Running free interactively
FREE MEMORY FROM TASK R2.4                                   reports the task size first.
Current task size: 32 (K bytes)
Enter new task size: 28                                      Reduce ourselves by 4K.
.LT
TASK  PAGE  TIME    TB    WS    PC    SR    BM    EM   CRU   PORT

*0/0   0     3    >7020 >719A >04E0 >1005 >7000 >E000 >0080 >000We are moved up to >7000.
.LM
FREE=4
.FREE 9                                                      Pass 9K bytes more through
.LT                                                          the command line.
TASK  PAGE  TIME    TB    WS    PC    SR    BM    EM   CRU   PORT

*0/0   0     3    >9420 >959A >04E0 >1005 >9400 >E000 >0080 >000Moving us up to >9400.
.LM
FREE=13
.RECOVER >6000                                               Finally we recover all.
.LT
TASK  PAGE  TIME    TB    WS    PC    SR    BM    EM   CRU   PORT

*0/0   0     3    >6020 >619A >04E0 >1005 >6000 >E000 >0080 >000We are back down to >6000.
.LM
FREE=0
._
```

13.21 FRMT303

Name: FRMT303
Function: Format 8" disks with TM990/303A.
Format: FRMT303

Systems: 101,102
Language: Assembly
Memory: 4K bytes

Restrictions: None

Description:

Format utilities.

A disk must be formatted before it can be used to store data. This simply means that all clock, address, and data marks as well as track and sector identification numbers must be recorded on the disk so that the controller can correctly find sectors of data.

Every different disk controller requires its own version of format. This also dictates the type of formatting to be done as well as the default values. PDOS looks at disk storage as a set of ordered, 256 byte sectors. Hence, cylinder, track, head, density, and other device dependent parameters are transparent to PDOS but must be handled by FORMAT and the disk read and write sector routines.

The FRMT303 utility formats 8" disks with the TM990/303A or 303A floppy disk controller board from Texas Instruments. Either single or double sided formatting can be done. FRMT303 prompts for the disk number (0-3), the number of disk sides (1 or 2) and the track(s) to format. A final verification is input by answering 'Y' to the prompt, and formatting begins.

A period is output for each track, any errors abort the process, and the 'SUCCESS' message indicates that the process completed without error. A ^C aborts the formatting and exits back to the PDOS monitor.

Examples:

```
.FRMT303
TM990/303A STANDARD FLOPPY FORMAT R2.4
LOGICAL UNIT=0
SIDES=1
TRACK=0,76
FORMAT UNIT 0, SECTORS 0 TO 76?Y
.....................................................................
SUCCESS!!
._
```

13.22 FRMT93

Name: FRMT93
Function: Formats 5" disks with ER3300.
Format: FRMT93

Systems: 101
Language: Assembly
Memory: 10K bytes

Restrictions: Only works single sided.

Description:

The FRMT93 utility formats 5" disks with a ER3300 floppy disk controller in a 101 PDOS system. This utility works with Western Digital's FD1793B-01 controller chip and issues a 'Write Track' command to format the disk. User options include logical unit, single or double density, 8 or 10 sectors per track, interleave, and tracks to format.

A period is printed each track and at the completion, the number of bytes written and the interleave table is printed. A ^C halts the formatting.

Examples:

```
.FRMT93
FORMAT 1793 DISK RZ.4
LOGICAL UNIT=0
DENSITY (S OR D)=DOUBLE
SINGLE DENSITY SECTORS/TRACK (8 OR 10)=8
INTERLEAVE=3
TRACK=0,39
DESTROY DISK?Y
................................
SUCCESS!
.
```

13.23 FRMTFDC1

Name: FRMTFDC1
Function: Formats 5" or 8" disks with the FDC-1.
Format: FRMTFDC1

Systems: STD
Language: Assembly
Memory: 16K bytes

Restrictions: None

Description:

The FRMTFDC1 utility formats disks with the FDC-1 from G W Three in the STD PDOS system. The FDC-1 uses the FD1793-B02 floppy controller, so that this utility is similar to FRMT93 for the ER3300 board. FRMTFDC1 reports the disk drive size that is present in the system and prompts for the logical unit number, the number of sides, the density format, the interleave factor, and the tracks. The user must then input a 'Y' to the FORMAT prompt, and the formatting begins. A period is output for each track, and upon completion, the number of bytes written on the last track and the interleave table are reported. A ^C aborts the formatting.

Examples:

```
.FRMTFDC1
FORMAT G W Three FDC-1 Disk, R2.4
  Drive size = 8"
LOGICAL DISK=0
SIDES (S OR D)=DOUBLE
DENSITY (S OR D)=DOUBLE
INTERLEAVE=1
TRACK=0,76<cr>
FORMAT DISK #0 ?Y
..........................................................................
SUCCESS!
LAST TRACK LENGTH=10399 BYTES (optimally 10416)
INTERLEAVE MAP:
1,14,2,15,3,16,4,17,5,18,6,19,7,20,8,21,9,22,10,23,11,24,12,25,13,26
.
```

13.24 FRMTVG

Name: FRMTVG
Function: Formats 5" disks with the Video Games SBC.
Format: FRMTVG

Systems: Video Games
Language: Assembly
Memory: 16K bytes

Restrictions: None, but the 9909 still has problems

Description:

The FRMTVG utility formats disks with the Video Games SBC The VG board uses the TMS 9909 floppy controller, so that this utility is unlike anything else. FRMTVG prompts for the logical unit number, the number of sides, the density format, and the tracks. No interleaving is currently supported. The user must then in put a 'Y' to the FORMAT prompt, and the formatting begins. A period is output for each track, and upon completion, a 'SUCCESS' message is printed. A ^C aborts the formatting.

Examples:

```
.FRMTVG
FORMAT Video Games Disk, R2.4
LOGICAL DISK=0
SIDES (S OR D)=DOUBLE
DENSITY (S OR D)=DOUBLE
TRACK=0,34<cr>
FORMAT DISK #0 ?Y
.................................
SUCCESS!
.-
```

13.25 FRMTW

Name: FRMTW
Function: Formats SASI disks with the ER3314 Host Adapter
Format: FRMTW

Systems: 101,102
Language: Assembly
Memory: 4K bytes

Restrictions: This utility has been known to destroy entire Winchesters full of data without mercy!

Description:

The FRMTW utility formats disks attached to the SASI compatible controller connected to the ER3314 Host Adapter. This utility sends either 'Format Track' or 'Format Disk' commands to the controller. Sector interleave is user selectable. If only one or more tracks are specified, instead of the entire disk, the FRMTW utility prompts for the number of sectors (or blocks) per track for the specified logical unit. This is needed by FRMTW since the 'Format Track' command requires a block number within the track you want to format. This allows formatting only a selected head and cylinder, if desired.

Beware! The Winchester DSRs address the main disk as LUN 0. There is a line in FRMTW:SR that is commented out that you might want to un-comment back in, which prohibits formatting logical unit 0. This would insure that you would never reformat the Winchester unintentionally. Be advised.

Examples:

```
.FRMTW
FORMAT ER3314 WINCHESTER R2.4
LOGICAL UNIT NUMBER=2
TRACK=<cr>COMPLETE DISK                    Hit <CR> for complete disk
INTERLEAVE=3
FORMAT LOGICAL UNIT 2?Y
SUCCESS!
.GO
FORMAT ER3314 WINCHESTER R2.4
LOGICAL UNIT NUMBER=2
TRACK=30,40                                Enter beg,end tracks
SECTORS/TRACK=26
INTERLEAVE=3
FORMAT LOGICAL UNIT 2?Y
.....
SUCCESS!
.
```

13.26 FRMTWS

Name: FRMTWS
Function: Formats SASI disks with the Micro/Sys Host Adapter
Format: FRMTWS

Systems: STD
Language: Assembly
Memory: 4K bytes

Restrictions: This utility has been known to destroy entire Winchesters full of data without mercy!

Description:

The FRMTWS utility formats disks attached to the SASI compatible controller connected to the Micro/Sys STD bus Host Adapter. This utility sends either 'Format Track' or 'Format Disk' commands to the controller. Sector interleave is user selectable. If only one or more tracks are specified, instead of the entire disk, the FRMTWS utility prompts for the number of sectors (or blocks) per track for the specified logical unit. This is needed by FRMTWS since the 'Format Track' command requires a block number within the track you want to format. This allows formatting only a selected head and cylinder, if desired.

Beware! The Winchester DSRs address the main disk as LUN 0. There is a line in FRMTWS:SR that is commented out that you might want to un-comment back in which prohibits formatting logical unit 0. This would insure that you would never reformat the Winchester unintentionally. Be advised.

Examples:

```
.FRMTWS
FORMAT STD WINCHESTER R2.4
LOGICAL UNIT NUMBER=2
TRACK=<cr>COMPLETE DISK                    Hit <CR> for complete disk
INTERLEAVE=3
FORMAT LOGICAL UNIT 2?Y
SUCCESS!
.GO
FORMAT STD WINCHESTER R2.4
LOGICAL UNIT NUMBER=2
TRACK=30,40                                Enter beg,end tracks
SECTORS/TRACK=26
INTERLEAVE=3
FORMAT LOGICAL UNIT 2?Y
.....
SUCCESS!
._
```

13.27 FSAVE

Name: FSAVE
Function: Restore files from links.
Format: FSAVE

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: None.

Description:

The FSAVE utility can be used to recover a file from a disk with a bad directory. FSAVE uses the forward/backward PDOS links to reconstruct a file.

The first prompt, 'DEST. FILE=', asks where the new file is to be stored. The second prompt, 'SOURCE DISK=', asks for the disk number where the file resides that is being recovered. The final prompt, '1st SECTOR', asks for the starting sector number. This number could be obtained from the DDUMP utility.

If the '1st SECTOR' has a valid backward link, then FSAVE asks if an attempt should be made to find the beginning of the file. If 'Y', then the backward links are followed until a null link is found. If 'N', then FSAVE begins with the specified sector. One sector at a time is read from the source disk and written to the new file. As each sector is transferred, the 'LINK' value is printed out.

The final link should be a 0. FSAVE cannot determine how many bytes are valid in the last sector. This must be done by the user with an editor.

FSAVE is exited with an <esc>.

```
.FSAVE
FILE RECOVERY R2.1
DEST. FILE=TEMP
SOURCE DISK=1
1st SECTOR=>E4
LINK=229
LINK=230
LINK=231
LINK=232
LINK=233
LINK=234
LINK=235
LINK=0
FILE RECOVERY R2.4
DEST. FILE=TEMP
SOURCE DISK=1
1st SECTOR=>E5
BACK LINK=230, FOLLOW? Y
BACK LINK=229
BACK LINK=228
BACK LINK=0
** BEGIN **
LINK=229
LINK=230
LINK=231
LINK=232
LINK=233
LINK=234
LINK=235
LINK=0
FILE RECOVERY R2.1
DEST. FILE=_
```

13.28 GLOBAL

Name: GLOBAL
Function: Globalize blocks of a 102 task's memory
Format: GLOBAL

Systems: 102
Language: Assembly
Memory: 4K bytes

Restrictions: None.

Description:

The GLOBAL utility sets or resets the global bits in the 102 PDOS task's memory map. This bit is used by PDOS to pass common memory blocks to spawned tasks. Any memory that is global to a task is also given to any tasks that are spawned by it.

Examples:

13.29 IMP

Name: IMP
Function: Install a memory page into 101 PDOS
Format: IMP
IMP <page #>

Systems: 101 w/ER3232 cards
Language: Assembly
Memory: 1K bytes

Restrictions: None.

Description:

The IMP utility installs a paged memory card, an ER3232 or equivalent, into the free memory pool of a 101 PDOS system. This is accomplished by setting the bits corresponding to the 32K byte block in the map area of PDOS. This memory is then available for task creation. If the page number of the ER3232 card, ranging from 1 through 7, is passed in the command line, then IMP assumes that the entire 32K bytes are populated on the board and the appropriate bits are set. If the utility is run with no parameters, then it prompts for page #, low address, and high address on the board. The bits are then set.

Examples:

```
.LM
FREE=0
.IMP 1                          Install a page 1 ER3232 card.
.LM
FREE=32                         Get 32K bytes more.
.IMP                            Install half a board.
INSTALL MEMORY PAGE R2.4
PAGE=2
LOW=>6000
HIGH=>A000
.LM
FREE=48
.
```

13.30 INIT

Name: INIT
Function: Initialize disks for PDOS file storage.
Format: INIT

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: Destroys all data on the disk.

Description:

Once a disk has been formatted, it must be initialized with a header sector, a sector bit map, and a file directory. Unlike the format utilities, INIT is independent of the disk controller.

INIT verifies that all specified sectors are usable by writing a null sector to each one. Hence, the disk is essentially cleared of data.

The parameters for INIT are:

1) disk number:	Ranges from 0 to 255.
2) number of sides:	Either 1 or 2.
3) media density:	Either S or D.
4) maximum number of directory entries or files:	Multiple of 8.
5) total number of sectors available on the disk:	Device/boot dependent.
6) a 16 character disk name:	You name it!

Most default to system standard values. The number of PDOS sectors can be set smaller than the actual total available thus allowing room for user defined storage. The maximum number of sectors for any one disk is 2^16 - 224 or 65312. A ^C aborts the initialization process and return the to PDOS monitor.

First, a sector allocation bit map image is constructed in memory, with all the sectors allocated. Each sector is then written to, and if an error is not returned, that sector is deallocated, or set as 'FREE', in the sector allocation bit map. Errors are reported to the console and those sectors remain allocated. After all sector numbers up to, but not including the Number of PDOS Sectors, then the header sector and other bit map sectors are written to the disk. If no errors occur, them the 'SUCCESS!!' message is printed to the console.

(13.30 INIT continued)

Examples:

```
.INIT
PDOS DISK INITIALIZATION R2.4
DISK #=^<cr>
Diskette:      SS:SD      SS:DD      DS:SD      DS:DD
  35 TRK 5"   184/280    360/552    368/560    728/1112
  80 TRK 5"   544/640   1080/1272  1088/1280  2168/2552
  76 TRK 8"   923/988   1846/1976  1846/1976  3692/3952
Bubble:
  TM990/210-3   156/252
Winchester:
  SA1002        8096/8192
  SA1004       16288/16384
  ST506        10002/10098
  ST412        20004/20100
DISK #=1
SIDES=2
DENSITY=0
FILES=256
SECTORS=3692
DISK NAME=PDOS INIT #1
INIT:   DISK #1
        DOUBLE SIDED
        DOUBLE DENSITY
        256 FILES
        3692 SECTORS
OK?Y
PLEASE WAIT.....
SUCCESS!!
._
```

13.31 INIT307

Name: INIT307
Function: Install the 307 ports into 101 PDOS
Format: INIT307

Systems: 101
Language: Assembly
Memory: 1K bytes

Restrictions: The system interrupt mask is not raised.

Description:

The INIT307 utility sets up PDOS to use a TM990/307 I/O card in the system. The interrupt mask of the CPU is not raised. The level 4 interrupt vector is copied into the level 8 interrupt vector, the level 8 interrupt mask bit is enabled at the system TMS 9901, the four port CRU addresses are set in the PDOS system table, and the TMS 9902s on the 307 are initialized.

User must 'IM 8' to raise mask.

The 307 CRU bases are set as follows:

9902 at base = >0600 assigns the 307 Port A to PDOS Port #3
9902 at base = >0680 assigns the 307 Port B to PDOS Port #4
9902 at base = >0700 assigns the 307 Port C to PDOS Port #5
9902 at base = >0780 assigns the 307 Port D to PDOS Port #6

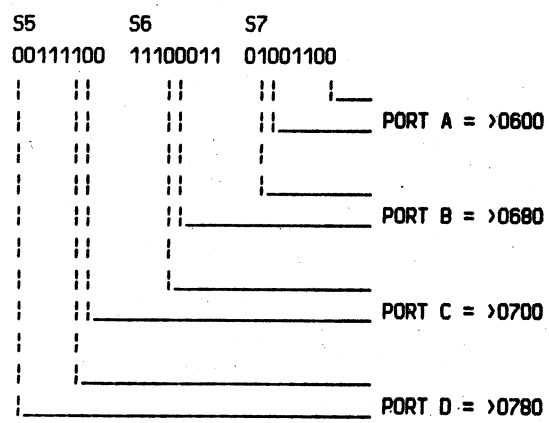
The TM990/307 switches must be set as follows:

1) Switch all TM990/307 ports for level 8 interrupts:

   For switch packs S1 through S4, turn switch 1 ON and 2-8 OFF. (See Figure 2-2 of 307 manual.)

2) Switch CRU base to >0600->0780:

   For switch packs S5, S6, S7, set:



13.32 JED

Name: JED
Function: Virtual screen editor
Format: JED
Restrictions: File size limited by disk size, system disk cannot be removed while running JED.

Systems: All
Language: Assembly
Memory: 16K bytes minimum

Description:

The JED utility is a virtual screen editor for the PDOS system. Buffers on the disk allow editing files larger than can be contained in system memory. Screen display is continuously updated, reflecting the actual file contents. This is a major system utility and is completely described in Chapter 11.

The following is a summary of the JED commands:

^A	INSERT FROM UP BUFFER	^U	COPY TO UP BUFFER
^B	BACKWARD SEARCH	^V	INSERT BYTE
^C	CANCEL	^W	WRITE FILE
^D	DEFINE MACRO	^X	TYPE AHEAD CANCEL
^E	EXECUTE MACRO	^Y	INSERT FILE
^F	FIND POINTER	^Z	BOTTOM OF BUFFER
^G	GET FILE	^\	DELETE BLOCK
^H	MOVE LEFT	^]	DELETE LINE -<CR>
^I	TAB	^^	DELETE LINE +<CR>
^J	MOVE DOWN	^_	DELETE RIGHT
^K	MOVE UP	RUBOUT	DELETE LEFT
^L	MOVE RIGHT	ESC ^A	LIST DIRECTORY
^M	CARRIAGE RETURN		
^N	NEW BUFFER	ESC ^E	INFINITE MACRO
^O	OUTPUT BLOCK	ESC ^H	JUMP LEFT
^P	PLACE POINTER	ESC ^I	INSERT MODE SELECT
^Q	QUIT	ESC ^J	JUMP DOWN
^R	RECENTER CURSOR	ESC ^K	JUMP UP
^S	SEARCH FORWARD	ESC ^L	JUMP RIGHT
^T	TOP OF BUFFER	ESC ^R	REPLACE MODE SELECT

Examples: See Chapter 11.

13.33 JEDY

Name: JEDY
Function: Screen editor (non-virtual)
Format: JEDY
Restrictions: File size limited by user memory.

Systems: All
Language: Assembly
Memory: 12k bytes minimum

Description:

The JEDY utility is a non-virtual screen editor for the PDOS system. It is identical to JED, except that no buffering is done out to the disk. Thus the system disk may be freely removed during editing with JEDY. In fact, JEDY can be used as a single drive file copier. Screen display is continuously updated, reflecting the actual file contents. This is a major system utility and is completely described in Chapter 11.

The following is a summary of the JEDY commands:

^A	INSERT FROM UP BUFFER	^U	COPY TO UP BUFFER
^B	BACKWARD SEARCH	^V	INSERT BYTE
^C	CANCEL	^W	WRITE FILE
^D	DEFINE MACRO	^X	TYPE AHEAD CANCEL
^E	EXECUTE MACRO	^Y	INSERT FILE
^F	FIND POINTER	^Z	BOTTOM OF BUFFER
^G	GET FILE	^\	DELETE BLOCK
^H	MOVE LEFT	^]	DELETE LINE -<CR>
^I	TAB	^^	DELETE LINE +<CR>
^J	MOVE DOWN	^_	DELETE RIGHT
^K	MOVE UP	RUBOUT	DELETE LEFT
^L	MOVE RIGHT	ESC ^A	LIST DIRECTORY
^M	CARRIAGE RETURN	ESC ^B	FREE BYTE COUNT
^N	NEW BUFFER	ESC ^E	INFINITE MACRO
^O	OUTPUT BLOCK	ESC ^H	JUMP LEFT
^P	PLACE POINTER	ESC ^I	INSERT MODE SELECT
^Q	QUIT	ESC ^J	JUMP DOWN
^R	RECENTER CURSOR	ESC ^K	JUMP UP
^S	SEARCH FORWARD	ESC ^L	JUMP RIGHT
^T	TOP OF BUFFER	ESC ^R	REPLACE MODE SELECT

Examples: See Chapter 11.

13.34 KILLM

Name: KILLM
Function: Kills all pending task messages.
Format: KILLM ‹task #›

Systems: All
Language: Assembly
Memory: 1K bytes

Restrictions: None.

Description:

The KILLM utility kills all pending messages of the specified task. The ‹task #› must be passed to KILLM, or an error is printed. This utility simply calls the XKTM primitive of PDOS.

Examples:

```
.SENDM
.SENDM 3,(HELLO #3, FROM #1)
.SENDM 4, HI MOM
.SENDM
*TASK #3: (HELLO #3, FROM #1)
*TASK #4:  HI MOM
.KILLM
PDOS ERR 67
.KILLM 3
.SENDM
*TASK #4:  HI MOM
.KILLM 4
.SENDM
._
```

13.35 LDIR

Name: LDIR
Function: Selective directory listings with wild cards.
Format: LDIR
LDIR {#}@:@;@/<disk #>{,<outfile>}

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: None.

Description:

The LDIR utility lists selected files from the disk directory. Wild cards and a condensed list are optional. The selection list is asked for if the program is just run, or it can be passed in the command line.

The file selection list consists of 4 fields: <file name> : <file extension> ; <directory level> / <disk #>. The <file name> consists of characters, single wild cards (*), or multiple wild cards (@). The <file extension> is selected in the same way. The <directory level> is a number from 0 to 254. If it is specified, then only files on that level are listed. Otherwise, all levels are included in the list.

The <disk #> specifies the PDOS disk of the source directory. <Disk #> defaults to the system disk. If an output file is specified, then the selected directory listing is written to the that file. The listing that is produced is identical to the PDOS system command 'LS'.

If a pound sign (#) precedes the selection list, then an abbreviated file listing is output. The format consists of the file name and extension, a semicolon, the file directory level, a comma, and the 'USED' file size. This is useful for creating long chain files (AC type) to do multiple assemblies or transfers.

(13.35 LDIR continued)

Examples:

```
.LDIR/1
LIST DIRECTORY R2.4
MASK=@:SR/1
OUTPUT=TEMP1
.SF TEMP1

DISK NAME=PDOS 2.4 #1/1                                  FILES=108/128
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 3   LPT:SR        TX         9/9      10:01 09/15/82  10:01 09/15/82
 3   NUL:SR        TX         2/2      10:01 09/15/82  10:01 09/15/82
 3   TTA:SR        TX        10/10     10:01 09/15/82  10:01 09/15/82
 3   TTO:SR        TX         9/9      10:01 09/15/82  10:02 09/15/82
 3   TTS:SR        TX         5/5      10:02 09/15/82  10:02 09/15/82
 6   AX95VG:SR     TX        51/51     10:02 09/15/82  10:03 09/15/82
 6   BOOT:SR       TX        89/89     10:03 09/15/82  10:04 09/15/82
 6   BOOTE:SR      TX         2/2      10:04 09/15/82  10:04 09/15/82
 6   BT210:SR      TX        16/16     10:05 09/15/82  10:05 09/15/82
 6   BT303:SR      TX        27/27     10:05 09/15/82  10:05 09/15/82
 6   BT3300:SR     TX        29/29     10:05 09/15/82  10:06 09/15/82
 6   BT3314:SR     TX        27/27     10:06 09/15/82  13:04 09/15/82
 6   BT95VG:SR     TX        25/25     10:06 09/15/82  10:07 09/15/82
 6   BTFLG:SR      TX         2/2      10:07 09/15/82  10:07 09/15/82
 6   RW303:SR      TX        27/27     10:07 09/15/82  10:07 09/15/82
 6   TRKDMP:SR     TX        22/22     10:08 09/15/82  10:08 09/15/82
.-

.LDIR/1 #BT@:SR/1,TEMP1
.SF TEMP1
DISK NAME=PDOS 2.4 #1/1                                  FILES=108/128V
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

BT210:SR;6,16
BT303:SR;6,27
BT3300:SR;6,29
BT3314:SR;6,27
BT95VG:SR;6,25
BTFLG:SR;6,2
.-
```

13.36 LEVEL

Name: LEVEL
Function: Short listing of directories sorted by level.
Format: LEVEL
LEVEL <disk #>{,<outfile>}

Systems: All
Language: Assembly
Memory: 1K bytes

Restrictions: None.

Description:

The LEVEL utility produces a short listing of a disk directory sorted by levels, and outputs it either to the console or to a file. The disk number and the optional output file can be passed to LEVEL in the command line. Otherwise, the utility prompts you for the disk # and the output file name (a <CR> directs the output to the console).

Examples:

```
.LEVEL/1
LIST DIRECTORY BY LEVELS R2.4
DISK #=1
OUTPUT=<cr>
DISK NAME=PDOS 2.4 #1/1
FILES=108/128
LEVEL FILES

 0    $LPT,$TTA,$TTO,$TTS,HLPTX,SY$STRT

 1    ALOAD,ASM,BACKUP,BFIX,BURN302,BURNP,COMP,DDMAP,DDUMP,DISCAT
      DISCVER,DNAME,DRGN,EDIT,ER3314,FDUMP,FREE,FRMT303,FRMT93,FRMTH
      FSAVE,GLOBAL,IMP,INIT,INIT307,JED,JEDD,JEDY,KILLM,LDIR,LEVEL
      LIBGEN,LINK,LOAD,LOGO,MDUMP,MEMTEST,MIAC,ORDIR,PRINT,RECOVER
      RENUMB,SENDM,SHOW,SORT,SYFILE,TERMINAL,TRANS,UPTIME,XBUG,XEROX

 3    LPT,LPT:SR,NUL:SR,TTA,TTA:SR,TTO,TTO:SR,TTS,TTS:SR

 5    ADV:DAT,ADVENT,AMAZING,BEAST,BLUFF,BOA,BULLS,CLOCK,CPOLY,DRAG
      EVAL,HANGMAN,HANOI,HPLOT,MERRY,OTHELLO,PUZZLE,ROBOT,SNOOPY,STARTREK
      WORM

 6    AX95VG,AX95VG:SR,BOOT,BOOT102,BOOT:SR,BOOTE,BOOTE:SR,BT210,BT210:SR
      BT303,BT303:SR,BT3300,BT3300:SR,BT3314,BT3314:SR,BT95VG:SR,BTFLG:SR
      DOBOOT,RW303:SR,TRKDMP,TRKDMP:SR
```

13.37 LIBGEN

Name: LIBGEN
Function: Generates a library for LINK from modules
Format: LIBGEN

Systems: All
Language: Assembly
Memory: 8K bytes + buffers

Restrictions: None.

Description:

The LIBGEN utility reads object code modules and generates a library file consisting of the module names and the names of all labels that are externally defined, or DEF'd out, by the module. No other label information is included. This file is then used by the LINK utility to optionally resolve any undefined references during the linking process.

Examples:

```
.LIBGEN
LIBRARY GENERATOR R2.4
OBJECT FILE=BT303
OBJECT FILE=BT3314
OBJECT FILE=BT210
OBJECT FILE=BT3300
OBJECT FILE=<cr>                                        No more files.
LIBRARY FILE=TEMP1/0
.FDUMP TEMP1/0                                          See XDLT01 thru XDLT04 DEF'd.
0000-000F  4254 3330 3300 0000 0000 0000 0000 0000  BT303...........
0010-001F  0001 5844 4C54 3031 4254 3333 3134 0000  ..XDLT01BT3314..
0020-002F  0000 0000 0000 0000 0001 5844 4C54 3032  ..........XDLT02
0030-003F  4254 3231 3000 0000 0000 0000 0000 0000  BT210...........
0040-004F  0001 5844 4C54 3033 4254 3333 3030 0000  ..XDLT03BT3300..
0050-005F  0000 0000 0000 0000 0001 5844 4C54 3034  ..........XDLT04
0060-006F  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ................
```

(13.37 LIBGEN continued)

```
.LINK                                                    Let's use them.
LINKER R2.4
*8,>F000                                                 Set base.
WAS >0000
*1,BOOT                                                  Load BOOT
*2                                                       See XDLTxx RED'd in by BOOT.
UNDEFINED DEF ENTRIES:
ISYS$  >0000  USER$  >0000  XDLT01 >0000  XDLT02 >0000  XDLT03 >0000
XDLT04 >0000
*13,TEMP1/O                                              Let library do it.
 1,BT303                                                 These messages printed by LINK.
 1,BT3314
 1,BT210
 1,BT3300
*2                                                       See them disappear.
UNDEFINED DEF ENTRIES:
ISYS$  >0000  USER$  >0000
*7                                                       Hoo Ray, it worked!
._
```

13.38 LINK

Name: LINK
Function: Link assembly and BASIC modules together.
Format: LINK

Systems: All
Language: Assembly
Memory: 12K bytes minimum

Restrictions: None.

Description:

The LINK utility combines object code modules and BASIC programs into a loadable file for user applications. External references are resolved and either a completed link or a partial link can be output. You have complete control over the relocatable bases, task numbers, code output format, and symbol status. This is a major PDOS utility for applications development and LINK is fully described in Chapter 11.

The following is a summary of the LINK commands:

COMMAND	DESCRIPTION
0,<FILE>	OPEN OUTPUT FILE
1,<FILE>	LINK FILE
2{,<FILE>}	LIST UNDEFINED REFS
3{,<FILE>}	LIST MULTIPLY DEFINED DEFS
4{,<FILE>}	LIST LINK MAP
5	OUTPUT PARTIAL LINK
6{,<ADR>}	OUTPUT OVERLAYS AND START TAG
7	EXIT TO PDOS
8{,<ADR>}	LIST/SET PSEG BASE ADDRESS
9{,<ADR>}	LIST/SET DSEG BASE ADDRESS
10	RESTART
11,<FL>,<S>,<P>	LOAD BASIC BINARY MODULE
12{,<M>}	0=NO DSEG, 1=NORMAL, 2=DSEG=>PSEG
13,<FILE>	LIBRARY
14{,DEFAULT}	SET DEFAULT TASK NUMBER
15{,FLAG}	OBJECT MODE, 0=OFF, 1=ON

Examples: See Chapter 11.

13.39 LOAD

Name: LOAD
Function: Load ASCII files from RS232 port to disk.
Format: LOAD

Systems: All
Language: Assembly
Memory: 1K bytes + buffer

Restrictions: File size limited to user memory.

Description:

The LOAD utility stores character data from the specified port into user memory and writes it to a PDOS file. Normally, port #2, the aux port on most PDOS systems, is used for load, but any port and baud rate can be entered. LOAD gets its characters from the PDOS buffer. To insure that characters are not dropped, no other tasks should be running or else the baud rate should be lowered.

LOAD first prompts for the port number and baud rate. Next it asks if you are 'READY'. If a 'Y' is entered, a long timeout begins waiting for the first character to be sent to the port from the RS232 source. If a 'N' is entered, LOAD immediately prompts with 'FILE='. Once characters begin to be received, the timeout becomes much shorter, so that the end of transmission can be sensed by LOAD in a reasonable time.

When LOAD is reentered, the file received is not destroyed and the prompt immediately asks for a file. This allows you to exit after the file has been sent, define the target file large enough, and then reenter LOAD to write out the buffer.

Examples:

```
.LOAD
PDOS FILE LOAD R2.4
PORT=2<cr>
BAUD=9600<cr>
READY?Y
LET'ER RIP!!
FILE=TEMP
.GO
PDOS FILE LOAD R2.4
READY?N
FILE=TEMP1
.
```

13.40 LOGO

Name: LOGO
Function: Load object code files, set checksums, and/or go.
Format: LOGO

Systems: All
Language: Assembly
Memory: 4K bytes + files

Restrictions: File size limited by user memory.

Description:

The LOGO or Load and GO utility allows TI9900 tag object code to be loaded into user memory, checksummed, written to disk in either binary or object code format, and then be moved down over the PDOS operating system for execution. This facilitates program development for standalone PDOS run modules where the interrupt and XOP vectors are required.

LOGO outputs a '*' as a prompt. A <CR> displays the following command menu:

```
 0{,adr}          BLWP @adr
 1,<file name>    LOAD FILE
 2,base           SET BUFFER BASE
 3,low,high,adr   SET CHECKSUM
 4,low,high,FILE  OUTPUT OBJECT
 5,low,high       WRITE BINARY TO DISK
 6                EXIT TO PDOS
BUFFER LIMITS ARE:
       LOW=>0000
      HIGH=>77EC
HIGHEST PC=>0000
*
```

0=BLWP @adr

The 0 selection moves the buffer data to its absolute address and executed a BLWP @<adr> instruction. This is the entry into the application program.

1=Load file

The 1 selection loads absolute object code files into the internal memory buffer. The addresses are supplied in the file and LOGO offsets them into the buffer base. Entry tags are reported, but otherwise ignored. IDT's are printed as they are read in.

2=Set buffer base

The 2 selection sets the logical base address of the internal LOGO buffer.

3=Set checksum

The 3 selection zeros location <adr>, sums the data from buffer address <low> up to address <high>, negates the result, and stores it in the buffer address <adr>. This sets a checksum in the application before it is written to disk or executed.

(13.40 LOGO continued)

The 4 selection writes an object code image of the data buffer from address <low> up to address <high> to the PDOS file <FILE>. Features 3 and 4 are used in generating the boot program EPROMs.

4=Output object to file

The 5 selection writes a binary image, without PDOS file links, from the data buffer address <low> up to address <high>. LOGO then prompts for the disk number and sector number. This is useful in writing out a new boot system or an application that boots into another type of system.

5=Write binary to disk

The 6 selection exits LOGO and returns to the PDOS monitor. This is used in chain files.

Exit to PDOS

LOGO follows the command menu with the current data buffer limits, which it calculates from the buffer base and the end of user memory. Finally, LOGO outputs the highest data buffer location loaded into by any of the modules.

Examples:

```
.LOGO
LOGO R2.4
*1,DEVPRGM1                                  Load module 1
LOADING.................
IDT='PRGM1   '
*1,DEVPRGM2                                  Load module 2
LOADING.................
IDT='PRGM1   '
*3,0,>1000,>80                               Set checksum
*5,0,>1000                                   Write binary to disk for future booting
DISK=1
SECTOR=1000
DONE!
*0,0                                         Go execute application
GO!!!!!!!!!!!!!!!!!!!!!

<Program executing>
```

13.41 MACRO

Name: MACRO
Function: Macro processor
Format: MACRO

Systems: All
Language: Assembly
Memory: 16K bytes minimum

Restrictions: None.

Description:

The MACRO utility is a macro processor whose operations include no operation, goto, let, pop, push, reset, end, and if. Special functions include number of elements in sublist N, a counter which increments each time used, nine parameters, sublists, substrings, and concatenation.

The program prompts for an input file list, an output file, a listing file, and an error file:

Calling sequence

```
.MACRO
IN=FILE1,FILE2,...
OUT=
LIST=
ERR=
```

The operations included in MACRO are listed below, along with their opcodes:

Opcodes

```
0 = NOP OR OPEN CODE
1 = GOTO
2 = LET
3 = POP
4 = PUSH
5 = RESET
6 = END
7 = IF
```

Special functions of the MACRO utility include the following:

Special functions

```
&@N     # of elements in sublist N
&#      Counter - increments each time used
&1-9    Parameters
&<exp>N Sublists
&"S,E"N Substrings
!       Concatenation
```

Examples:

See the separate documentation furnished with the MACRO utility.

13.42 MDUMP

Name: MDUMP
Function: Dump memory block to console or file.
Format: MDUMP
MDUMP <beg>,<end>{,<outfile>}

Systems: All
Language: Assembly
Memory: 2K bytes

Restrictions: None.

Description:

The MDUMP utility dumps system memory to an output file or user console. The dump includes a hex and ASCII image of the memory. The output can be halted and continued with the space bar.

Examples:

```
.MDUMP
MEMORY DUMP R2.4
OUTPUT=
START=>6000
END=>6100
6000-600F  0000 6175 6026 004B 0000 600C 6200 6300  ..au`&.K..`.b.cP
6010-601F  635E 6020 OFFF 62BA 0000 2D97 6220 170A  c^` ..b:..-.b ..
6020-602F  3E36 3230 3000 0042 3000 4101 3C42 0000  >6200..B=.A.<B..
6030-603F  4101 8042 0A0D 4102 0642 0000 3200 0045  A..B..A..B..2..E
6040-604F  4355 5445 5320 494E 2043 4C4B 5753 2057  CUTES IN CLKWS W
6050-605F  4F52 4853 5041 4345 0D2A 0D2A 0920 5231  ORKSPACE.*.*. R1
6060-606F  203D 2053 495A 4520 4F46 2041 5353 454D   = SIZE OF ASSEM
6070-607F  424C 5920 5052 4F47 5241 4D53 0D2A 0952  BLY PROGRAMS.*.R
6080-608F  3131 203D 2042 4547 494E 4E49 4E47 204F  11 = BEGINNING O
6090-609F  4620 4153 5345 4D42 4C59 2050 524F 4752  F ASSEMBLY PROGR
60A0-60AF  414D 530D 2A09 5231 3320 3D20 4E45 5720  AMS.*.R13 = NEW
60B0-60BF  574F 524B 5350 4143 450D 2A09 5231 3420  WORKSPACE.*.R14
60C0-60CF  3D20 4F4C 4420 5354 4154 5553 2042 4C4F  = OLD STATUS BLO
60D0-60DF  4348 0D2A 0952 3135 203D 2053 5441 5455  CK.*.R15 = STATU
60E0-60EF  5320 5245 4749 5354 4552 0D2A 0D41 4C4C  S REGISTER.*.ALL
60F0-60FF  4434 3009 4D4F 5620 5239 2C52 3132 0D09  D40.MOV R9,R12..
START=0
END=30
0000-000F  2FDC 00CC 2278 2298 226C 228C 2FDC 0390  /\.L"x"."l"./\..
0010-001F  2F60 059C 2F60 059C 2F60 059C 2260 2280  /`../`../`.."`".
START=
.
```

13.43 MEMTEST

Name: MEMTEST
Function: Random memory tester.
Format: MEMTEST

Systems: All
Language: Assembly
Memory: 1K bytes

Restrictions: Destroys PDOS system memory.

Description:

The MEMTEST utility writes random data from address 0000 up to the specified end address and then reads it back to verify it. This test is identical to the option 100 in the BOOT:SR program. MEMTEST prompts for the end address to test and then the test proceeds to write through memory. This destroys PDOS as we know it. A period and bell is output at the end of each pass. Errors are listed as the address followed by the exclusive 'OR' of the data read and the data written.

Examples:

```
.MEMTEST
END ADDR=>E000
GO.............
.....
ERROR:ADR,DATA=4FFE,0040
...
```

13.44 MIAC

Name: MIAC
Function: Memory inspect and change, with block copy.
Format: MIAC

Systems: All
Language: Assembly
Memory: 1K bytes

Restrictions: None.

Description:

The MIAC utility is a general memory inspect and change utility, with both a memory block dump and a block copy features. MIAC is identical the to 101 option in the BOOT:SR program.

Using the memory inspect and change, system memory can be examined, altered, or copied. Both a hexadecimal and an ASCII dump is output. There is no prompt. When the cursor is the extreme left, any one of the following three modes can be invoked.

Inspect and Change

To examine and alter a memory location, input one hex number followed by a carriage return. The location address is output, followed by a colon and the contents of the location. Entering a hex number alters that location. A space bar increments the location by 2 and the next location's contents are displayed for alteration. A minus (-) decrements the location and a control C cancels any input. An escape exits to the PDOS monitor. Entire words are altered, so no single byte change is possible.

Memory Dump

To examine a block of memory, input two hex numbers, separated by a space or comma. The memory contents from the first address through the second address displays to the terminal in both a hexadecimal format, and in an ASCII format (ignoring the uppermost bit). If the character represented by a byte is not printable (i.e. less that >20), then a period is printed in its place. A space bar momentarily halts the output and then restarts the display. An escape returns to the PDOS monitor.

(13.44 MIAC continued)

Memory Copy

To copy one block of memory into another, input three hex numbers, separated by single spaces or commas. The memory contents from the first address through the second address copies into a block starting at the third address. The copy mode uses a move byte (MOVB) instruction so that odd address boundaries are possible. This can be used as a memory test. Remember that PDOS resides from >0000 through >2FFF and that BASIC resides from >3000 through >5FFF, so be careful.

Examples:

```
.MIAC/1
MEMORY IAC R2.4
8000: 2020 1234                                          Enter IAC
8000: 1234 +                                             Alter >8000 and >8002
8002: 2020 4321
8002: 4321 -
8000: 1234
8000 8010                                                Dump block
8000:  1234 4321 2020 2E69 6E20 7468 6520 636F  .4C!  .in the co
8010:  6D6D 616E 6420 6C69 6E65 2E20 2020 2020  mmand line.
80 100                                                   Dump PDOS
0080:  A55A FFFF 0317 6200 2FDC 1786 1770 007D  %Z....b./\...p.}
0090:  AA00 9B3D 0078 2E64 0080 0180 0E00 0A00  *..=.x.d........
00A0:  0A40 0A80 0AC0 0B00 0019 0001 000A 0014  .@...@..........
00B0:  5359 2453 5452 5400 0A0D 072E 001A 0034  SY$STRT........4
00C0:  0068 00D0 01A0 0340 04D0 0638 4B00 257F  .h.P. .@.P.8K.%.
00D0:  12BE 095D 04AC 0253 0126 0067 2F64 3C00  .>.].,.S.&.g/d<.
00E0:  2F62 3C00 2F60 1800 2F68 FF01 2F66 0D01  /b<./`../h../f..
00F0:  2F6A 6400 201D 201F 201F 2020 1F20 1F20  /jd. . . .  . .
0100:  0B10 05C1 0621 2F50 1505 E800 2F4E C861  ...A.!/P..h./NHa
8000 8020 8001                                           Copy byte >12
8000 8020
8000:  1212 1212 1212 1212 1212 1212 1212 1212  ................
8010:  1212 1212 1212 1212 1212 1212 1212 1212  ................
8020:  1212 2020 2020 2020 2020 2020 2020 2020  ..
<ESC>                                                    Return to PDOS
.
```

13.46 PSPELL

Name: PSPELL
Function: Check spelling of ASCII files.
Format: PSPELL

Systems: All
Language: Assembly
Memory: 24K bytes minimum

Restrictions: Dictionary size limited to user memory.

Description:

The PSPELL utility is a spelling checker. PSPELL compares words of two or more letters to its internal dictionary, presents those words that it cannot find to the console, and prompts the user for the desired action. The dictionary is read in from the disk file D$DICT and the following menu is printed:

```
PSPELL PROCESSOR R2.3e
QUIT.....0
CHECK....1
COMPOSE..2
CREATE...3
DUMP.....4
DELETE...5
OPTION=_
```

The 0 selection reports the vital statistics of the dictionary and asks if you want to update the D$DICT file with the current contents in memory. Control then returns to the PDOS monitor.

0=QUIT

The 1 selection checks an existing PDOS file for unknown words. PSPELL prompts for an input file, and then prompts for various output options. First is the output file. This file is the corrected input file. The second file is the LIST file, which is the input file with line numbers. If a line is corrected, it prints both the old line and the altered line with a 'C' to indicate the correct line. The third prompt is for the ERROR file. This file is numbered corresponding to the LIST file, but contains only the erroring lines and their corrections. The prompts look like this:

1=CHECK file

```
OPTION=1
IN=CH13:01
OUT=
LIST=
ERR=TEMP/0
```

Lines are then read from the input file and the words are checked. When a word is not found, it is presented to the console for user action, with the prompt:

```
This line has an errror in it.
     < errror >              Change, Insert, List, Pass? _
```

(13.45 ORDIR continued)

```
.LS
DISK NAME=PAUL #1MD/0                                    FILES=48/64
LEV  NAME:EXT      TYPE     SIZE     DATE CREATED    LAST UPDATE

 1   ASM           OB       74/74    02:00 12/16/80  02:01 12/16/80
 1   BURNP         OB       14/15    08:06 01/07/81  13:02 01/20/81
 1   DDUMP         OB        8/9     02:10 12/16/80  20:22 06/01/32
 1   DFIX          OB        3/3     11:18 02/20/81  20:09 02/26/81
 1   DO            AC        3/3     02:08 12/16/80  02:08 12/16/80
 1   DO$           AC        1/1     02:52 12/16/80  02:52 12/16/80
 1   DRGN          OB        8/8     02:09 12/16/80  02:09 12/16/80
 1   EDIT          OB       16/16    02:12 12/16/80  02:13 12/16/80
 1   FDUMP         OB        5/5     16:20 01/16/81  14:59 01/27/81
 1   FSAVE         OB        4/4     02:13 12/16/80  02:13 12/16/80
 1   JEDY          OB       42/42    02:01 12/16/80  02:01 12/16/80
 1   LIST                    6/10    02:08 12/16/80  02:16 12/16/80
 1   LOGO          OB        5/5     02:14 12/16/80  02:14 12/16/80
 1   TRANS         OB       12/13    02:15 12/16/80  20:02 02/24/81
.
```

13.45 ORDIR

Name: ORDIR
Function: Alphabetizes and compresses disk directory.
Format: ORDIR
ORDIR <disk #>

Systems: All
Language: Assembly
Memory: 4K bytes minimum

Restrictions: This rewrites directory sectors, errors may destroy file information.

Description:

The ORDIR utility reorganizes and alphabetizes a disk directory. All directory sectors are scanned, ordered, and then rewritten to the disk. If errors occur while trying to write out the directory sectors, ORDIR prompts you for an alternate sector number to write the directory to. then using the alter mode of DDMAP, you can reconstruct the directory later (possibly after reformatting the header track).

Examples:

```
.LS
DISK NAME=PAUL #1MD/0                                  FILES=48/64
LEV  NAME:EXT      TYPE     SIZE     DATE CREATED    LAST UPDATE

 1   TRANS         OB       12/13   02:15 12/16/80  20:02 02/24/81
 1   BURNP         OB       14/15   08:06 01/07/81  13:02 01/20/81
 1   ASM           OB       74/74   02:00 12/16/80  02:01 12/16/80
 1   DFIX          OB        3/3    11:18 02/20/81  20:09 02/26/81
 1   LOGO          OB        5/5    02:14 12/16/80  02:14 12/16/80
 1   DDUMP         OB        8/9    02:10 12/16/80  20:22 06/01/32
 1   DO$           AC        1/1    02:52 12/16/80  02:52 12/16/80
 1   FDUMP         OB        5/5    16:20 01/16/81  14:59 01/27/81
 1   DO            AC        3/3    02:08 12/16/80  02:08 12/16/80
 1   ORGN          OB        8/8    02:09 12/16/80  02:09 12/16/80
 1   EDIT          OB       16/16   02:12 12/16/80  02:13 12/16/80
 1   FSAVE         OB        4/4    02:13 12/16/80  02:13 12/16/80
 1   JEDY          OB       42/42   02:01 12/16/80  02:01 12/16/80
 1   LIST                    6/10   02:08 12/16/80  02:16 12/16/80
.ORDIR/1
ORDER DISK DIRECTORY R2.4
DISK=0
.........REWRITE DIRECTORY
```

(13.46 PSPELL continued)

Legal responses are:

'C' for changing the word in an edit mode.
'I' for inserting the word into the dictionary.
'L' for listing the entire line again.
'P' for inserting the word into a pass table.
<CR> for ignoring the word.
<ESC> for exiting the current file check.

The word edit mode is just like insert mode in JEDY. When you have fixed the offending word, type <CR> and PSPELL checks the line all over again. The pass table is a temporary table which is built each time PSPELL is run, words passed to this table do not appear again as errors, but are not inserted into the dictionary either. If you ignore the word with <CR> then no action is taken and PSPELL moves on. At the end of the file or if you hit <ESC>, PSPELL reports:

```
TOTAL LINES=24
TOTAL WORDS=103
PASS TABLE SIZE=11
DICTIONARY WORDS=4275
AVAILABLE BYTES=13066
UPDATE DICTIONARY?_
```

This includes the total number of lines and words processed from the last input file, the number of bytes in the pass table, the internal dictionary size, and the number of bytes still available for either dictionary or pass table words. PSPELL then prompts to update the D$DICT file. A 'Y' writes out the file to the disk.

2=COMPOSE file

The 2 selection acts just like the 0 option, except that the input comes directly from the keyboard. All the output options are keyed in and things are just like above.

3=CREATE dictionary

The 3 selection clears the current internal dictionary and prompts for a file. This file is then read into the buffer memory as the new dictionary, with no questions asked. This file contains complete words that comprise the new 'correct' word list. The file is possible an edited version of the file output by the 4 option.

4=DUMP dictionary

The 4 selection unpacks the internal dictionary and outputs it to a file for viewing or editing.

5=DELETE word

The 5 selection deletes the specified word from the dictionary, if it is found.

(13.46 PSPELL continued)

Examples:

```
.PSPELL
PSPELL PROCESSOR R2.4
QUIT.....0
CHECK....1
COMPOSE..2
CREATE...3
DUMP.....4
OPTION=1
IN=TEMP1
OUT=
LIST=
ERR=ERROR1

PDOS is a powerful multi-user, multi-tasking operating
system developped by Eyring Research Institute, Inc., for the
     < developped >          Change, Insert, List, Pass? C
system developed by Eyring Research Institute, Inc., for the
     < Inc >                 Change, Insert, List, Pass? P
Texas Instruments compatible provessor family.
     < provessor >           Change, Insert, List, Pass? C
Texas Instruments compatible processor family.
This development software is designed for scientific, educational,
industrial, and business applications.

PDOS consists of a small, real-time, multi-tasking kernel layered by
file management, floaling point, and user monitor modules.
     < floaling >            Change, Insert, List, Pass? C
file management, floating point, and user monitor modules.
The 2k byte kernel provides sychonization and control of events
     < sychonization >       Change, Insert, List, Pass? C
The 2k byte kernel provides sychronization and control of events
     < sychronization >      Change, Insert, List, Pass? C
The 2k byte kernel provides synchronization and control of events
occurring in a real-time environment using semaphores, events,
messages, mailboxes, and suspension primitives.
All user console I/O as well as pther useful conversion
     < pther >               Change, Insert, List, Pass? C
All user console I/O as well as other useful conversion
and housekeeping routines are included in the PDOS kernel.
TOTAL LINES=14
TOTAL WORDS=129
PASS TABLE SIZE=3
DICTIONARY WORDS=4475
AVAILABLE BYTES=24865
UPDATE DICTIONARY?N
```

(13.46 PSPELL continued)

```
PSPELL PROCESSOR R2.4
QUIT.....0
CHECK....1
COMPOSE..2
CREATE...3
DUMP.....4
OPTION=0
.SF ERROR1
     C system developped by Eyring Research Institute, Inc., for the
   3   system developed by Eyring Research Institute, Inc., for the
     C Texas Instruments compatible provessor family.
   4   Texas Instruments compatible processor family.
     C file management, floaling point, and user monitor modules.
   9   file management, floating point, and user monitor modules.
     C The 2k byte kernel provides sychonization and control of events
     C The 2k byte kernel provides sychronization and control of events
  10   The 2k byte kernel provides synchronization and control of event
     C All user console I/O as well as pther useful conversion
  13   All user console I/O as well as other useful conversion
.
```

13.47 RECOVER

Name: RECOVER
Function: Recovers task memory from FREE and ALOAD.
Format: RECOVER
RECOVER <lowaddr>

Systems: All
Language: Assembly
Memory: 8K bytes

Restrictions: The 102 portions of RECOVER are much different from the other PDOS systems.

Description:

The RECOVER utility performs valuable memory management functions in the PDOS system. These functions include recovering memory adjacent to the beginning of a task to get more memory and moving the task down into that memory. RECOVER allows you to drop the BASIC interpreter and get 12K bytes more for application programs. In the 102 PDOS system, the BASIC interpreter can even be restored by RECOVER without re-booting the entire system.

101, STD, VG systems

In all PDOS systems except 102 PDOS, you simply pass to RECOVER a new low address for your task. This cannot be lower than >3000, since that is where PDOS resides. If the address is from >3000 to >5FFE, the a constant is checked to see if BASIC is still resident. If it is, then the prompt, 'DROP BASIC?', is output. A 'Y' reply drops the BASIC interpreter and sets the new task low memory at the specified address. If BASIC has already been dropped from the system, then the task is moved down without any operator prompting. The RECOVER utility comes in handy for regaining the memory lost when the FREE or the ALOAD utilities have been run, eating up your low memory.

102 system

In the 102 PDOS system, recover is much more useful. RECOVER reads the contents of the memory mapper and prints the physical location of your task at present. Next, the available memory blocks are read from PDOS memory and output to the console. Then, the first question asked deals with BASIC. If BASIC is present, as obviated by the map contents, RECOVER asks if you simply want to drop it. Type a 'Y' to drop BASIC and the utility asks for which available pages you would like to replace BASIC with. Just enter the hex numbers from the previously printed available page list, and you are moved down over BASIC (even though BASIC is still really there).

Drop BASIC

(13.47 RECOVER continued)

If BASIC is not present, then RECOVER asks if you simply want to restore it. Type a 'Y' and the utility restores BASIC, moves you up (if needed), frees up your previous memory, and then exits to PDOS. Simple, eh?

Restore BASIC

Typing a 'N' to the BASIC prompt drops into a common routine for memory manipulation that is quite confusing. 102 PDOS users need to experiment with this feature, look at the RECOVER:SR listing, and take good notes things progress.

Examples:

```
.RECOVER/1
MEMORY RECOVERY R2.4
CURRENT TASK MEMORY LIMITS:
      LOW = >6000
     HIGH = >E000
NEW LOW LIMIT=>3000
DROP BASIC?Y                                          BASIC is in, so we are asked.
NEW TASK MEMORY LIMITS:
      LOW = >3000
     HIGH = >E000
.LT
TASK  PAGE  TIME    TB    WS    PC    SR    BM    EM   CRU   PORT

*0/0   0     3    >3020 >319A >04E0 >1005 >3000 >E000 >0080 >0001
.FREE/1 12                                            Move back up above BASIC.
.LT
TASK  PAGE  TIME    TB    WS    PC    SR    BM    EM   CRU   PORT

*0/0   0     3    >6020 >619A >04E0 >1005 >6000 >E000 >0080 >0001
.RECOVER/1 >3000                                      Now no BASIC: no prompt.
.LT
TASK  PAGE  TIME    TB    WS    PC    SR    BM    EM   CRU   PORT

*0/0   0     3    >3020 >319A >04E0 >1005 >3000 >E000 >0080 >0001
._
```

13.48 RENUMBER

Name: RENUMBER
Function: Renumbers BASIC programs
Format: RENUMBER
Restrictions: Program must be in ASCII format 'EX'.

Systems: All
Language: BASIC
Memory: 16K bytes + buffer

Description:

RENUMB renumbers a PDOS BASIC program. Undefined transfers are tagged and any warnings listed.

Examples:

.SF MERRY

```
 10  PRINT @"C"
 15  K=40
 25  ST=1: $CH="*": GOSUB 320
 100  FOR I=0 TO 77
 105   IF I>36: IF I<42: GOTO 120
 110   PRINT @[23,I];"_";
 120  NEXT I
 135  ST=INT[RND*20]+1: X=RND*127: IF X<31: GOTO 135
 140  $CH=%X: GOSUB 320
 150  IF MF
 152    THEN PRINT @[22,21];"MERRY CHRISTMAS";
 155    THEN PRINT @[22,43];"TURKEY NECK";@[29,29]: MF=0
 160    ELSE PRINT @[22,21];"               ";
 165    ELSE PRINT @[22,43];"          ";: MF=1
 190  FOR IT=1 TO 50
 200   X=INT[RND*23]: Y=INT[RND*78]
 210   IF Y>40-X: IF Y<38+X: GOTO 200
 220   PRINT @[X,Y];".";
 230   X=INT[RND*23]: Y=INT[RND*76]
 240   IF Y>38-X: IF Y<38+X: GOTO 230
 250   PRINT @[X,Y];"   ";
 260  NEXT IT
 275  GOTO 135
 320  FOR I=1 TO 19 STEP ST
 330   FOR J=1 TO 2*I-1
 340    PRINT @[I+1,39-I+J];$CH;
 350   NEXT J
 360  NEXT I
 370  FOR I=21 TO 23
 380   PRINT @[I,37];"¦   ¦";
 390  NEXT I
 400  RETURN
```

(13.48 RENUMBER continued)

```
.RENUMBER

* RENUMBER *    11/25/80
NAME OF INPUT FILE: MERRY
NAME OF OUTPUT FILE: LIST1/5


                    MERRY              LIST1/5
                    ===============    ===============
LOW LINE # =        10                 10
  RANGE 1             10 ... 400         10 ... 400       32 LINES
HIGH LINE # =       400                400
                    TOTAL NUMBER OF LINES = 32

1) RENUMBER OUTPUT LINES
2) LIST OF MULTIPLY DEFINED LINE NUMBERS
3) WRITE OUT NEW LINE NUMBERS TO OUTPUT FILE
4) RESTORE ORIGINAL LINE NUMBERS TO OUTPUT LINES
5) HELP

SELECT OPTION: 1

                    MERRY              LIST1/5
                    ===============    ===============
LOW LINE # =        10                 10
  RANGE 1             10 ... 400         10 ... 400       32 LINES
HIGH LINE # =       400                400
                    TOTAL NUMBER OF LINES = 32

RENUMBER FROM LINE: 100
TO LINE: 319
STARTING WITH A NEW LINE NUMBER OF: 100
WITH INCREMENTS OF: 5

                    MERRY              LIST1/5
                    ===============    ===============
LOW LINE # =        10                 10
  RANGE 1             10 ... 110         10 ... 110       6 LINES
  RANGE 2             120 ... 275        115 ... 195      17 LINES
  RANGE 3             320 ... 400        320 ... 400      9 LINES
HIGH LINE # =       400                400
                    TOTAL NUMBER OF LINES = 32

1) RENUMBER OUTPUT LINES
2) LIST OF MULTIPLY DEFINED LINE NUMBERS
3) WRITE OUT NEW LINE NUMBERS TO OUTPUT FILE
4) RESTORE ORIGINAL LINE NUMBERS TO OUTPUT LINES
5) HELP
```

(13.48 RENUMBER continued)

```
SELECT OPTION: 1

                    MERRY               LIST1/5
                    ===============     ===============
LOW LINE # =        10                  10
  RANGE 1             10 ... 110          10 ... 110        6 LINES
  RANGE 2             120 ... 275         115 ... 195       17 LINES
  RANGE 3             320 ... 400         320 ... 400       9 LINES
HIGH LINE # =       400                 400
                    TOTAL NUMBER OF LINES = 32

RENUMBER FROM LINE: 320
TO LINE: 500
STARTING WITH A NEW LINE NUMBER OF: 500
WITH INCREMENTS OF: 5

                    MERRY               LIST1/5
                    ===============     ===============
LOW LINE # =        10                  10
  RANGE 1             10 ... 110          10 ... 110        6 LINES
  RANGE 2             120 ... 400         115 ... 540       26 LINES
HIGH LINE # =       400                 540
                    TOTAL NUMBER OF LINES = 32

1) RENUMBER OUTPUT LINES
2) LIST OF MULTIPLY DEFINED LINE NUMBERS
3) WRITE OUT NEW LINE NUMBERS TO OUTPUT FILE
4) RESTORE ORIGINAL LINE NUMBERS TO OUTPUT LINES
5) HELP

SELECT OPTION: 2


*** END OF MULTIPLY DEFINED LINE NUMBERS ***
HIT <CR> TO CONTINUE <cr>

                    MERRY               LIST1/5
                    ===============     ===============
LOW LINE # =        10                  10
  RANGE 1             10 ... 110          10 ... 110        6 LINES
  RANGE 2             120 ... 400         115 ... 540       26 LINES
HIGH LINE # =       400                 540
                    TOTAL NUMBER OF LINES = 32

1) RENUMBER OUTPUT LINES
2) LIST OF MULTIPLY DEFINED LINE NUMBERS
3) WRITE OUT NEW LINE NUMBERS TO OUTPUT FILE
4) RESTORE ORIGINAL LINE NUMBERS TO OUTPUT LINES
5) HELP
```

(13.48 RENUMBER continued)

```
SELECT OPTION: 3

 10  PRINT @"C"
 15  K=40
 25  ST=1: $CH="*": GOSUB 500
 100  FOR I=0 TO 77
 105   IF I>36: IF I<42: GOTO 115
 110   PRINT @[23,I];"_";
 115  NEXT I
 120  ST=INT[RND*20]+1: X=RND*127: IF X<31: GOTO 120
 125  $CH=%X: GOSUB 500
 130  IF MF
 135    THEN PRINT @[22,21];"MERRY CHRISTMAS";
 140    THEN PRINT @[22,43];"TURKEY NECK";@[29,29]: MF=0
 145    ELSE PRINT @[22,21];"               ";
 150    ELSE PRINT @[22,43];"           ";: MF=1
 155  FOR IT=1 TO 50
 160   X=INT[RND*23]: Y=INT[RND*78]
 165   IF Y>40-X: IF Y<38+X: GOTO 160
 170   PRINT @[X,Y];".";
 175   X=INT[RND*23]: Y=INT[RND*76]
 180   IF Y>38-X: IF Y<38+X: GOTO 175
 185   PRINT @[X,Y];"   ";
 190  NEXT IT
 195  GOTO 120
 500  FOR I=1 TO 19 STEP ST
 505   FOR J=1 TO 2*I-1
 510    PRINT @[I+1,39-I+J];$CH;
 515   NEXT J
 520  NEXT I
 525  FOR I=21 TO 23
 530   PRINT @[I,37];"¦   ¦";
 535  NEXT I
 540  RETURN


1) RENUMBER OUTPUT LINES
2) LIST OF MULTIPLY DEFINED LINE NUMBERS
3) WRITE OUT NEW LINE NUMBERS TO OUTPUT FILE
4) RESTORE ORIGINAL LINE NUMBERS TO OUTPUT LINES
5) HELP

SELECT OPTION:
```

13.49 RS232

Name: RS232
Function: Slave driver for an RS232 pseudo-disk device.
Format: CT RS232,2

Systems: All
Language: Assembly
Memory: 2K bytes

Restrictions: Task accesses disks external to PDOS. Careful not to define files from slave and master systems at the same time.

Description:

The RS232 utility is a link routine to run in a 'slave' PDOS system to allow disk access by a 'master' PDOS system. The 'master' system has the BT232 Device Service Routine in its boot program. The 'master' communicates, at 9600 baud, with the disk devices on the 'slave' over an RS232C line that is swapped. This communication protocol is defined in the BT232:SR listing and is very simple. The RS232 utility configures the TMS 9902 at the port 2 CRU base address at 9600 baud, with 8 character data. Thus bytes can be sent, rather than just 7-bit characters, speeding the communications. Faster baud rates might be successful if interrupts are disabled, too.

Examples:

After connecting the 'master' and 'slave' port 2 with a swapped RS232 cable, (only pins 2, 3, and 7 needed), the RS232 task is started in the 'slave':

```
.CT RS232,2
TASK #1
._
```

Then the boot program on the 'master' system is initiated and the RS232 device is selected:

```
SELECT RS232 ? Y
...
```

Now the disk devices 0-3 in the 'slave' system can be accessed by the 'master' system as disks 8-11, even for booting:

```
?8
BOOTED!
HIT RETURN
PDOS STD R2.4
MN,DY,YR=_
```

13.50 SENDM

Name: SENDM
Function: Send a message to a task.
Format: SENDM
SENDM <task #>,<message>

Systems: All
Language: Assembly
Memory: 1K bytes

Restrictions: Messages are at most 50 characters long.

Description:

The SENDM utility sends messages to other tasks and lists all pending messages to the console. If the <task #> parameter is included, then the message following the delimiter (comma or blank) is placed in a PDOS message buffer, if any are available. Messages can be up to 50 characters in length and can be sent to any task, even your own task. When the destination task returns the the PDOS monitor, then the message is printed to that task's port and the message buffer is cleared.

If no <task #> parameter is included, then SENDM outputs all pending task messages to the console.

Examples:

```
.SENDM
.SENDM 3,(HELLO #3, FROM #1)
.SENDM 4, HI MOM
.SENDM
*TASK #3: (HELLO #3, FROM #1)
*TASK #4:  HI MOM
.KILLM
PDOS ERR 67
.KILLM 3
.SENDM
*TASK #4:  HI MOM
.KILLM 4
.SENDM
._
```

13.51 SHOW

Name: SHOW
Function: Show a file, with inter-file movement and print.
Format: SHOW
SHOW <file>

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: None.

Description:

The SHOW utility displays a file, regardless of size, to the terminal. SHOW provides for movement within the file and for printing parts of the file out to unit 2. The display is screen oriented and the bottom row of the screen is used for operator input and file status outputs.

The commands for file movement in SHOW are:

^T	Move to top of file.
^Z or ^B	Move to end of file
CR	Move down 1 screen full.
^K	Move up 1 block.
LF	Move down 1 block.

The block size defaults to 100 bytes and can be changed with one of the other user commands listed below:

+nnn	Set block size = nnn bytes.
-nnn	Set block size = -nnn bytes.
Pnnn	Send nnn lines from HERE out Unit 2.
^C	Halt printing.
ESC	Exit to PDOS.

To use the printer option, you must set up the unit 2 base and baud rate before running SHOW. These commands are entered and echoed in the lower lefthand corner of the display. The count (nnn) is any decimal number. Note that using the (-) command reverses the sense of the <LF> and ^K movement keys.

On the bottom line of the display SHOW reports the current byte index of the character in the upper lefthand corner of the screen, along with the byte index of the END-OF-FILE marker. A continuously updating clock is displayed in the lower righthand corner when SHOW is waiting for your input.

13.52 SORT

Name: SORT
Function: A multi-key file sort routine.
Format: SORT

Systems: All
Language: Assembly
Memory: 8K bytes + index

Restrictions: None.

Description:

The SORT utility sorts files on multiple keys, with records of either fixed or variable length. One SORT option produces a file of 4-byte indices for use in other applications. SORT provides information about the number of records, as specified, that it can process in the memory available. An internal index system is employed on the first pass, and then the actual sorting occurs on the file using these indexes. Large files are sorted faster if they are contiguous, since random access is used during the sort. Either ascending or descending order is selectable.

The SORT utility first prompts for and input and an output file. It next asks for the index file name. The index output file is an option and consists of the sorted 4-byte key indices. There values can be used as inputs to other applications of your own design.

SORT then asks if the records to be sorted are of fixed or variable length. A 'F' response indicates that the records are of fixed length. The record length must then be input, and the program then begins asking for the key definition.

A 'V' response indicates that the records are of variable length and the character that delimits the records must then be input. The character is input as a hexadecimal byte constant, preceded by a '>'. To sort files of text, the records are of variable length and delimited with a carriage return (>0D). Entering a <CR> to the record type prompt, instead of 'F' or 'V', selects both variable record mode and <CR> delimiter.

Next, you define the keys that SORT is to use. These are input as two parameters: key index and key length. The key index ranges from 1 to the maximum record length and the key length indicates the number of characters to use in sorting by that key. SORT continues prompting for keys until a <CR> is entered. The total key length is reported, along with the total number of records that can be sorted in memory. The message, 'PLEASE WAIT', then output, the input file is read through, and the index table is built. During the sort, the keys are used in the order that they were input. Next the message, 'SORT COMPLETE', is output and the output file is written sequentially by accessing the input file according to the indices.

(13.52 SORT continued)

Finally, the number of records sorted is reported and the program returns control to the PDOS monitor.

Examples:

```
.SORT                                              Let's sort a source file
PDOS SORT R2.4
INPUT FILE=RS232:SR
OUTPUT FILE=TEMP1/0
INDEX FILE=<cr>                                    No index file
(F)IXED OR (V)ARIABLE? V                           Could have just hit <CR>
RECORD DELIMITER=>00
KEY INDEX=1                                        Only alphabetical
KEY LENGTH=10
KEY INDEX=<cr>
TOTAL KEY LENGTH=16
RECORDS AVAILABLE=1934
(A)SCENDING OR (D)ESCENDING? D                     Do it backwards
PLEASE WAIT.....
SORT COMPLETE...
141 RECORDS SORTED
.SF TEMP1/0                                        Let's see it
WRTE2   BL @GETC        ;GET CHARACTER
WRTE    BL @GETS
WAIT4   XSWP            ;SWAP WHILE WAITING
WAIT2   MOV @CRUB,R12
WAIT    STWP R13
START   STST R15        ;CAPTURE STATUS
SENDOK  LI R0,'.'*256   ;DATA OK!
...
         JMP ERROR
          JGT WRTE2     ;N
          JGT READ02    ;N
          JGT ER100     ;N
          JEQ WRTE      ;Y
          JEQ WAIT      ;Y
          JEQ INIT
.
```

(13.52 SORT continued)

Now, to see that fixed length record sorting doesn't work too well on text files, we sort the same file another way:

```
.SORT
PDOS SORT R2.3e
INPUT FILE=RS232:SR
OUTPUT FILE=TEMP1/O
INDEX FILE=<cr>
(F)IXED OR (V)ARIABLE? F                          Select fixed length
RECORD LENGTH=5                                   5 byte/record
KEY INDEX=1                                       Add some silly indices
KEY LENGTH=3
KEY INDEX=5
KEY LENGTH=1
KEY INDEX=4
KEY LENGTH=1
KEY INDEX=<cr>
TOTAL KEY LENGTH=12
RECORDS AVAILABLE=2579
(A)SCENDING OR (D)ESCENDING? A                    In upward order
PLEASE WAIT.....
SORT COMPLETE...
.SF TEMP1/O
                        *                         See the garbage?
               *
              ;CH     09/O    ;O,1    ;ACK    ;BYT    ;DON    ;DON
;INI    ;N
               ;N
               ;N,     ;REA    ;SEN    ;SEN    ;TIM    ;Y,     B *R
... (garbage)
.
```

13.53 SYFILE

Name: SYFILE
Function: Convert an object code file to compressed code.
Format: SYFILE
SYFILE <objfile>,<sysfile>

Systems: All
Language: Assembly
Memory: 1K bytes

Restrictions: None.

Description:

The SYFILE utility creates a system file (type=SY) from a TMS 9900 tag object file (type=OB). All unnecessary tags and control characters are discarded. Only tags 2 (start tag), 9 (absolute address), A (relocatable address), B (data), and C (relocatable data) are output to the new file. Also, the four byte value field is reduced to a single 16-bit binary number. A system file occupies one third less space on a disk and loads proportionally faster.

Examples:

```
.ASM LOGO:SR,TEMP
ASM R2.4
SRCE=LOGO:SR
OBJ=TEMP
LIST=
ERR=
XREF=

END OF PASS 1
0 DIAGNOSTICS
END OF PASS 2
0 DIAGNOSTICS
.LS 0
DISK NAME=PAUL #27MD/0                          FILES=43/64
LEV  NAME:EXT      TYPE     SIZE     DATE CREATED    LAST UPDATE

 0   TEMP          OB        6/6    10:34 02/27/81  12:40 03/16/81
.SYFILE
SYFILE R2.4
OBJECT FILE=TEMP
SYSTEM FILE=LOGO
.LS
DISK NAME=PAUL #27MD/0                          FILES=43/64
LEV  NAME:EXT      TYPE     SIZE     DATE CREATED    LAST UPDATE

 1   LOGO          SY        4/4    15:26 02/27/81  12:40 03/16/81
._
```

SYFILE can be spawned into an implied background task with two file names as arguments.

```
.@SYFILE FILE:OB,FILE
```

13.54 TERMINAL

Name: TERMINAL
Function: Set terminal cursor functions for task only.
Format: TERMINAL
TERMINAL <type char>

Systems: All
Language: Assembly
Memory: 4K bytes

Restrictions: none.

Description:

The TERMINAL utility sets the position cursor and clear screen codes in the task status block. This facilitates using various types of terminals on the same PDOS system. Each task has its own characters for these two functions, which are initialized, when the task is started, to the characters set by BFIX. TERMINAL provides an easy way for a task to change its function characters while the system is running.

If a legal <type character> is passed in the command line, then TERMINAL simply enters the corresponding sequences into the user status block. Otherwise, the utility prints the following table of options:

```
FIX TASK TERMINAL R2.4
TERMINALS:
    A=ADDS REGENT 25
    D=DECSCOPE (VT52 or VT100)
    H=HAZELTINE 1520
    I=INTERTUBE II
    L=LEAR SEIGLER
    S=SOROC
    U=USER DEFINED
TYPE=_
```

and prompts the user for an input. Enter the letter representing the type of terminal you are using, if listed. If your terminal is not listed, enter a 'U'. Then simply enter the hexadecimal representation of the sequences used by your peculiar terminal, surrounding each with angle brackets. One or two characters are acceptable.

(13.54 TERMINAL continued)

Examples:

```
.TERMINAL/1                                        Invoke interactively
FIX TASK TERMINAL R2.4
TERMINALS:
    A=ADDS REGENT 25
    D=DECSCOPE (VT52 or VT100)
    H=HAZELTINE 1520
    I=INTERTUBE II
    L=LEAR SEIGLER
    S=SOROC
    U=USER DEFINED
TYPE=H                                             Try Hazeltine
.TERMINAL
FIX TASK TERMINAL R2.4
TERMINALS:
    A=ADDS REGENT 25
    D=DECSCOPE (VT52 or VT100)
    H=HAZELTINE 1520
    I=INTERTUBE II
    L=LEAR SEIGLER
    S=SOROC
    U=USER DEFINED
TYPE=U                                             Let's look at them now...
CLEAR SCREEN CHARACTERS.....<ESC><5C>   NEW=
POSITION CURSOR LEAD CHARACTERS..<ESC><11>   NEW=
.TERMINAL/1 S                                      Reset to Soroc
.GO                                                Examine again
FIX TASK TERMINAL R2.4
TERMINALS:
    A=ADDS REGENT 25
    D=DECSCOPE (VT52 or VT100)
    H=HAZELTINE 1520
    I=INTERTUBE II
    L=LEAR SEIGLER
    S=SOROC
    U=USER DEFINED
TYPE=U
CLEAR SCREEN CHARACTERS.....<ESC>*      NEW=
POSITION CURSOR LEAD CHARACTERS..<ESC>=      NEW=
.
```

13.55 TRANS

Name: TRANS
Function: Selective file transfer, with wild cards.
Format: TRANS
TRANS @:@;@/<source disk #>,<dest. disk #>{select string}

Systems: All
Language: Assembly
Memory: 12K bytes minimum

Restrictions: None.

Description:

The TRANS utility transfers one or more files from one disk to another. Wild cards, date limits, and query are provided for greater flexibility. This utility requires the system to have at least two disk drives. TRANS is useful in reconstructing a fractured disk into a disk where all files are contiguous and any unused sectors are recovered.

Files are transferred according to a source selection list, which is the first prompt or parameter. This list consists of five fields: <file name> : <file extension> ; <directory level> / <disk #> / <options>. The <file name> consists of characters, single character wild cards (*), or multiple character wild cards (@). The <file extension> is specified the same way. The <directory level> is either a number from 0 to 254 or an (@), indicating all file levels. If no <directory level> option is specified, then it defaults to an (@). If a number is selected, then only files on that level are considered for a transfer. Note that level 255 is illegal.

The <disk #> specifies the PDOS disk of the source directory. <Disk #> defaults to the system disk. The <option> field is optional and selects the FROM date, the TO date, or the QUERY options. The format of FROM is '/FMN/DY/YR', where 'MN' is a month, 'DY' is a day, and 'YR' is the year. Only source files whose date of last update is newer, or more recent, are considered in the transfer. In a similar fashion, the TO date option is specified by '/TMN/DY/YR', and limits the transfer to those files whose last update occurred before or on MN/DY/YR. The QUERY option is specified in the source selection list by '/Q' and causes TRANS to ask whether you want each eligible file transferred or not. Answer 'Y' for yes and 'N' for no. These options can come in any order.

(13.55 TRANS continued)

The second prompt or parameter is the destination list, consisting of a <disk #> and <options>. The <disk #> is required and tells TRANS where to files are going. The <options> are optional and consist of single letters, separated by '/' characters. Legal destination list characters and there meaning are as follows:

D = Transfer files that are defined on the dest.
F = Don't delete files before transfer to them.
N = Only transfer files that are newer.
O = Override protection flags on dest. files.
U = Transfer files that are undefined on the dest.

The default options are equivalent to '/D/U', but if any destination options are specified, the defaults are reset. This means that for a FAST transfer with some files already defined, you need to use the '/D/U/F' options. These options can be mixed in any order.

The files in the directory of the source disk is read and compared to the entire selection specification. If the file qualifies on all counts, then the file name, level, and size is printed to the console. The operator is prompted for 'Y' or 'N' if the QUERY option was selected. Files to be transferred are first deleted from the destination disk, unless the F (fast) option was selected. Next, a new file is defined. If the new file cannot be defined contiguously then the message '** FRACTURED FILE' is printed and TRANS sizes the destination disk to see if there is enough space there for the whole file. If there is enough room, then the transfer continues. If not, an error message is printed and [illegible] asked whether or not to continue transferring other files.

Finally, the file and its attributes are transferred to the destination disk. If an error occurs during the transfer, TRANS asks if the transfer should to continue with the other files.

A driver (files beginning with '$') is transferred by altering the '$' to a lower case 'z', transferring the file, and renaming both the original and the copy with the '$'. A ^C will stop the transferring of programs after the current transfer is completed.

(13.55 TRANS continued)

Examples:

```
.TRANS
TRANS R2.4
SOURCE=^<cr>
     SOURCE = FILE:EXT;LEVEL/DISK{/OPTIONS}
              /Q => QUERY
 /FROM 10/01/81 => BEGINNING DATE
   /TO 10/31/81 => END DATE
DESTINATION = DISK{/OPTIONS}
              /D => DEFINED
              /F => FAST
              /N => NEWER UPDATE
              /O => OVERRIDE FLAGS
              /U => UNDEFINED
SOURCE=<esc>
.TRANS T@:@/O/Q,1/D/F/U
    TEMP;1                    2          TRANSFER?Y
    TEMP1;1                   8          TRANSFER?N
    TRANS;1        SY     **  15          TRANSFER?Y
WRITE ERROR=106
CONTINUE?Y
.
```

13.56 XBUG

Name: XBUG
Function: PDOS resident system debugger, with single step.
Format: ALOAD XBUG

Systems: All
Language: Assembly
Memory: 8k bytes

Restrictions: None.

Description:

The XBUG debugger is an assembly language debugger which is loaded into memory using the ALOAD utility. The first location is the entry address. When first entered, XOP vector 12 is altered for breakpoints. Four breakpoints are available as well as single stepping, memory dumps, and various inspect and change modes. XBUG is a major system development utility that is fully described in Chapter 11. The following is a summary of XBUG commands:

Breakpoint XOP = 12

```
XBUG R2.4
?
A {PC}                  Assemble
B {#}{adr}              Set/clear breakpoints
C from,to,into          Copy memory block
D from{,to}             Disassemble
E base                  Sent disassembly base
F from,to,data{+}       Find data
G {PC}{,WS}{,SR}        Go execute program
L FILE                  Load FILE from disk
M {adr}{,adr}           Memory I/C
Q {base},{bits}         CRU I/C
P {PC}                  Set program counter
R {#}                   Register I/C
S                       Single Step
U {unit}                Set unit
W {WP}                  Set workspace pointer
X                       Exit to PDOS w/breaks
Y {SR}                  Set status register
Z                       Exit to PDOS w/o breaks
```

Examples: See Chapter 11.

13.57 XEROX

Name: XEROX
Function: Single disk drive file copier.
Format: XEROX

Systems: All
Language: BASIC
Memory: 12K bytes minimum

Restrictions: None.

Description:

The XEROX utility is used to copy a file on one disk to a file on another - even if only one drive exists in the system. XEROX is an PDOS BASIC program.

XEROX first prompts for the source file. It then reads as much as possible into user memory. The user is prompted to change disks and enter a destination file name. If a <CR> is entered, it is assumed that the new file name is the same as the old file name. If a file already exists, it is first deleted and a new file defined. The data is then written to the disk.

If the file is too large to fit in user memory, the source and destination disks may have to be swapped several times in order to copy the file. XEROX prompts the user when to swap disks and which disks to use.

Examples:

```
.XEROX
XEROX R2.4
SOURCE FILE=BURNP
...READING FILE...
INSERT DESTINATION DISK.  FILE=BURNP
...WRITING FILE...
*** ALL DONE ***
XEROX R2.4
SOURCE FILE=BURNP
...READING FILE...
INSERT DESTINATION DISK.  FILE=TEMP/1
...WRITING FILE...
*** ALL DONE ***
XEROX R2.4
SOURCE FILE=
.
```