CHAPTER 8

DISK DEVICE SERVICE ROUTINES

This chapter explains the DSR format, the environment in which it operates, and how to alter the parameters in the standard DSRs to customize them to your system's needs. In addition, the following discussion examines the features of the generalized boot program, called BOOT:SR, gives a detailed example of how to write a completely new DSR, and tells how to LINK and burn a new Boot EPROM.

(CHAPTER 8 DISK DEVICE SERVICE ROUTINES continued)

8.1 INTRODUCTION

Hardware independent

PDOS uses read and write sector primitives to interface to secondary storage devices. This hardware independence allows for floppy or Winchester disks, magnetic tapes, bubble memories, external RAM or even another computer as candidates for 'disks' in PDOS. No modification of the operating system itself is required and all PDOS features remain unchanged. This is accomplished by using EPROM resident Device Service Routines (DSRs).

Interfaces through R/W sector

Device Service Routines

A DSR is a software module that is external to the operating system, and contains a specific set of entry points, parameter tables, and linker references. The DSR is required to perform a defined set of functions with regard to a particular storage device. If some functions are not needed, the DSR must gracefully ignore the operating system's calls. However all defined entries must be incorporated into the DSR (e.g., device initialization or motor off.)

DSRs in assembly language

DSRs are written in assembly language and linked to the general boot program module, BOOT:SR, using the LINK utility. This process might seem overwhelming at the outset, but you should remain calm and read this chapter carefully. It is helpful to get a listing of one of the standard DSRs, such as BT303:SR or BT3314:SR, and look for the various features as they are discussed. Don't despair; someone weaker than yourself has successfully written a DSR. The process is a bit complicated, but the rules are well defined and complete.

8.2 BOOT:SR

All PDOS systems, regardless of the hardware, use a boot program residing in EPROM for power-up start. One assembly language module is the basis for the EPROMs in all PDOS systems. The source code for the program is a file called BOOT:SR. This module contains the start-up code for each PDOS system, the boot program itself, and all the necessary linkage to PDOS. The following sections describe all of the entry addresses in the boot program and the system definition flags, which come from a file called BTFLG:SR. A general discussion of the boot program's features is followed by an explanation of the available system boot options. Subroutines in the boot program are externally defined for incorporation into the optional user written EPROM routines.

The size of BOOT:SR is less than 2K bytes, depending on the system flags. It must be included in all boot EPROMs and be linked first. The start-up code, if any, is located at the beginning of the ROM, and the major portion of the program resides up from >F800.

8.2.1 BOOT ENTRY ADDRESSES

The PDOS boot EPROMs contain read and write logical sector routines, called Device Service Routines (DSR), and a system boot program. Entry addresses are at address >F800 and include controller initialization and motor off routines. Other functions of the boot EPROMs consist of a memory test, a memory inspect/change utility, make boot facilities, and entry into user EPROM program.

```
Read and write logical sector
System boot
Memory inspect and change
Memory test
```

The read and write sector DSRs are the link between PDOS and secondary storage devices. Reference to a 256 byte sector is by disk number (R0), logical sector number (R1), and buffer address (R2). Errors are returned in register R0. These errors are device dependent, range from 100 to 32768, and are defined by the individual device service routines.

```
R0=disk #
R1=logical sector
R2=buffer address
```

(8.2.1 BOOT ENTRY ADDRESSES continued)

Memory addresses >F000 through >FFFF are reserved for EPROM routines. The boot EPROMs for a TM990/101 system reside at memory addresses >F800 through >FFFF. PDOS 102 and STD are mapped at address >F000 through >FFFF. All use entry points located at address >F800 through >F81F. These are defined as follows:

>F800 READ LOGICAL SECTOR. XRSE and XRSZ primitives pass R0, R1, and R2 to this routine. (See 5.2.11 READ SECTOR.)

>F804 WRITE LOGICAL SECTOR. XWSE passes R0, R1, and R2 to this routine. (See 5.2.23 WRITE SECTOR.)

>F808 INITIALIZE LOGICAL SECTOR. XISE passes R0, R1, and R2 to this routine. Initialize sector is equivalent to write sector except that no PDOS ID check is made on the header sector. (See 5.2.7 INIT SECTOR.)

>F80C INITIALIZE CONTROLLER. This routine is called once via a 'BL' instruction before PDOS system initialization. Device dependent initialization procedures are handled here.

>F810 MOTOR OFF ROUTINE. This routine is called once every second via a 'BL' instruction and is for controller devices, which need constant attention. Such is the case with 5" mini-floppies, which require the motor to be turned off after a period of inactivity.

```
        AORG >F800
BOOTV   B @XRSEZ        ;READ SECTOR
        B @XWSEZ        ;WRITE SECTOR
        B @XISEZ        ;INITIALIZE SECTOR
        B @XDITC        ;INIT CONTROLLER
        RT              ;MOTOR OFF (IGNORED)
```

(8.2.1 BOOT ENTRY ADDRESSES continued)

>FFFC LOAD VECTOR. This address in the EPROMs contains the cold start-up vector addresses used by the boot ROMs. This address is only required by the 101 PDOS, since the 101 CPU card must map the EPROMs at >F800.

>0000 RESET VECTOR. This address is needed by those PDOS systems that map the EPROM at address >0000 on power-up. These systems subsequently map RAM at location >0000 and the EPROM is mapped high at >F000. The start-up code that does the map flipping resides at the beginning of the EPROM, with the RESET vector locations >0000 and >0002 containing the address of the boot program.

Note that the >F800 addresses above are actually at >E800 on the 9995 SBC from Video Games since the I/O devices on that board are mapped at >F800. The start-up code for the Video Games board copies the EPROM up into RAM at >E000, and the boot program then executes from that location.

8.2.2 BTFLG:SR - SYSTEM FLAGS

Each file, including BOOT:SR, should include the boot flags during assembly. Executing a 'COPY BTFLG:SR' in the source code accomplishes this. Thus, every file knows whether or not the system has switches, what CPU card is used, and which devices are present. File 'BTFLG:SR' is listed to the right. A description of each flag follows:

FLG101 Setting this flag to a 1 configures the assembly of BOOT:SR and the DSR modules for a TM990/101MA system. A zero means that the EPROMs are for another system.

FLG102 Setting this flag to a 1 configures the assembly of BOOT:SR and the DSR modules for a TM990/102 system. A zero means that the EPROMs are for another system.

FLG95S Setting this flag to a 1 configures the assembly of BOOT:SR and the DSR modules for a G H Three 9995/STD system. A zero means that the EPROMs are for another system.

FLG95V Setting this flag to a 1 configures the assembly of BOOT:SR and the DSR modules for a Video Games 9995 system. A zero means that the EPROMs are for another system.

Note that exactly one of the above four flags must be set to a 1 and the other three set to zero in order for the BOOT:SR assembly to work.

```
*       BTFLG:SR        09/20/82
*****************************************
*       SYSTEM CONFIGURATION FLAGS
*****************************************
*
FLG101  EQU 1   ;TM990/101M
FLG102  EQU 0   ;TM990/102
FLG95S  EQU 0   ;STD TMS9995
FLG95V  EQU 0   ;VIDEO GAMES TMS9995
*
FLGROM  EQU 0   ;0=RAM, 1=EPROM
FLGAB   EQU 0   ;0=AUTO-BAUD, 1=AUTO-BOOT
FLGSW   EQU 1   ;0=NO SWITCHES, 1=SWITCHES
LDV1    EQU 1   ;LOGICAL DEVICE #1 (XDLT01)
LDV2    EQU 1   ;LOGICAL DEVICE #2 (XDLT02)
LDV3    EQU 0   ;LOGICAL DEVICE #3 (XDLT03)
LDV4    EQU 0   ;LOGICAL DEVICE #4 (XDLT04)
*
NOFF    EQU 0   ;DISK OFF DEVICE #
*
FLG95   EQU FLG95S!FLG95V
DVSEL   EQU LDV4*2+LDV3*2+LDV2*2+LDV1
```

(8.2.2 BTFLG:SR - SYSTEM FLAGS continued)

FLGROM	It is possible to burn into EPROM the complete 101 PDOS system and have it reside LOW, at address >0000. In this system, the DSRs are addressed at >2800 and the on-board RAM is mapped at >F000 for system variables. When assembling the R/H sector routines from BOOT:SR for an EPROM 101 PDOS, set this flag the a 1. IN ALL OTHER CASES, this flag must be zero.	FLGROM = 0 = Normal boot system 1 = PDOS in EPROM
FLGAB	Setting this flag to a 1 configures the assembly of BOOT:SR to auto-boot the PDOS system into memory without user intervention. A zero means that the EPROMs are to auto-baud the port and wait for your selections. ZERO IS THE NORMAL VALUE FOR THIS FLAG.	FLGAB = 0 = Normal boot procedure 1 = Always auto-boot
FLGSW	Setting this flag to a 1 configures the assembly of BOOT:SR for a system that reads sense switches. A zero means that the EPROMs are for a system that has no switches, such as the STD PDOS system. With no switches, the boot program goes through the select device sequence described later in this chapter. Only the standard 101 PDOS EPROMs are configured for sense switches (FLGSW=1); ALL OTHER PDOS SYSTEMS SET FLGSW TO ZERO.	FLGSW = 0 = No switches 1 = Switches
LDV1 LDV2 LDV3 LDV4	These four flags correspond to the four devices supported by BOOT:SR. Setting any of these flags to a 1 configures the assembly of BOOT:SR to assume that the corresponding DSR is to be linked into the EPROM and that the XDLT<#> is defined. This means that the DSR is in the EPROM and NOT that the device is necessarily installed in the system. Determining the latter is done during the running of the program by the device select logic (either switches or prompts). A zero means that the corresponding DSR is not to be linked into the EPROMs. These flags are only used to conditionally assemble in one of the four device initialize calls nothing too drastic.	LDV1 = 0 = Disks 0-3 not installed 1 = Disks 0-3 installed LDV2 = 0 = Disks 4-7 not installed 1 = Disks 4-7 installed LDV3 = 0 = Disks 8-11 not installed 1 = Disks 8-11 installed LDV4 = 0 = Disks 12-99 not installed 1 = Disks 12-99 installed

(8.2.2 BTFLG:SR - SYSTEM FLAGS continued)

NOFF — Setting this flag to a number from 1 to 4 configures the assembly of BOOT:SR to call the Disk Off entry of the corresponding DSR module. The standard boot program can only call this entry for one device, but more can be supported by altering BOOT:SR itself. A zero means that none of the devices needs to be serviced by the Disk Off routine while PDOS is running.

```
NOFF = 0 = No disk off routine required
       1 = Disks 0-3 required disk off service
       2 = Disks 4-7 required disk off service
       3 = Disks 8-11 required disk off service
       4 = Disks 12-99 required disk off service
```

FLG95 — If this flag is a 1 then the EPROMs are for a PDOS system using a TMS 9995 CPU. A zero means that the EPROMs are for a TMS 9900 system. This flag simply tells BOOT:SR where to locate the workspace for the boot program: at >7000 for 9900 systems or at >F080 for 9995 systems (in the on-chip RAM). This flag is automatically defined and should not be altered.

```
FLG95 = 0 = TMS 9900 system
        1 = TMS 9995 system
```

DVSEL — This flag is a combination of the logical device flags with the devices bit encoded. It is currently not used by BOOT:SR, but is made available for use in other modules.

```
DVSEL = %0000
         \\\\___ Disks 0-3 installed
          \\\___ Disks 4-7 installed
           \\___ Disks 8-11 installed
            \___ Disks 12-99 installed
```

8.2.3 BOOT PROGRAM

The bootstrap program begins executing on the 101MA via the LOAD vector at memory address >FFFC. All other PDOS systems, the 102, STD and Video Games versions, begin executing the boot program via the RESET vector at memory address >0000. There is no way to map EPROM at address >FFFC in these systems. If auto-boot has been either selected by a switch (101MA CPU and 102 with 307 for switches) or burned into the boot program EPROM (FLGAB = 1), then the program proceeds to boot the system from the lowest installed disk device. Otherwise, the program waits for a character on the main CPU port.

TM990/101MA = >FFFC
TM990/102, STD = >0000

The main workspace used by the boot program is at location >7000 for the PDOS systems using the TMS 9900. The TMS 9995 based PDOS systems locate the main workspace in the on-chip RAM at location >F080. Various parameter locations follow the main workspace and a secondary workspace, used for calling the R/W sector routines while booting, is located 64 bytes (>40) above the beginning of the main workspace (at >7040 and >F0C0, respectively). This is followed by a buffer, whose address is externally defined for use by the DSRs and user routines.

Main workspace = >7000 (or >F080 for 9995)

Secondary workspace = >7080 (or >F0C0 for 9995)

The first character entered is timed and used to set the baud rate of the main console port. This is referred to as auto-bauding a port. A carriage return auto-bauds all ports correctly.

Auto-baud main port

9902 initialized for 11 bits:
1 start bit
7 bit character
1 even parity
2 stop bits

After the port has been bauded, the boot program tests the Data Set Ready (DSR) signal in the main port. If no DSR is present the boot program prints an error message to alert you that PDOS will not output to the terminal.

Test DSR high
DSR LOW

If a system initialization subroutine is present in the EPROMs, the boot program calls it, ISYS$, using a Branch and Link instruction. Otherwise the boot program proceeds to sum memory from >F800 to >FFFA. If this sum is nonzero, then a 'CHECKSUM ERROR' message is reported, indicating that something has happened to the EPROMs and could be a source of problems. The checksum is set by the LOGO utility before the EPROMs are burned. The location that is altered for the checksum is >F812.

Call ISYS$, if present

Checksum EPROM
CHECKSUM ERROR

(8.2.3 BOOT PROGRAM continued)

Next, you may be queried as to which storage devices are installed. This occurs only when there are no configuration switches available, such as with a TM990/102 or SBC95/1 system. A single 'Y' character selects the device. Anything else ignores the device, even though the DSR is linked into the EPROM.

Finally, the PDOS boot menu is displayed. You may now select from various functions including:

<CR>,0-99	System boot from disk numbers 0 through 99. Boot sector constants within the EPROMs select the correct default sector of the boot. Auto-boot or a carriage return selects the lowest numbered storage device installed. See DEVICE SELECTION.	<LOAD vector> <carriage return> SELECT TM990/303? Y {Optional} SELECT ER3314? {Optional} *PDOS BOOT R2.4 0-99=BOOT 100=MEMORY TEST 101=IAC 102=BOOT 103=MAKE BOOT 104=AUX ? System boot
100	System memory test (100). A pass is made through memory writing random data and then a second pass verifies memory content.	Memory test
101	Memory inspect and change (101). System memory is examined, altered, or copied. Both a hexadecimal and an ASCII dump is provided.	Memory inspect and change
102,103	OTHER BOOT (102) and MAKE BOOT (103) routines. A system boot from any logical sector is done by the OTHER BOOT routine. The MAKE BOOT routine writes memory from >0000 to >6000 to any disk # beginning at any logical sector number.	Boot and make boot
104	AUX programs (104). Other routines are placed in the lower half of the EPROM space (>F000 through >F7FF) and called via an entry from the externally defined label USER$.	Auxiliary program
105	Go to PDOS (105). Although not in the menu message, entering a 105 executes a BLWP @>0000. This is provided so that a boot that is aborted may be continued without rebooting.	GOOO

(8.2.3 BOOT PROGRAM continued)

The following flow diagram indicates the main features of the boot program.

	101MA	102	STD	VG
1. Initiate boot program with EPROM at address:	LOAD >F800	RESET >0000	RESET >0000	RESET >0000
Set map information in:	--	'612	--	LDCR
Copy EPROM high	--	--	--	MOV
Swap EPROM high with:	--	CKON	SBO	LDCR
EPROM at address:	--	>F000	>F000	>E000

2. Go to BAUD routine

   Auto booting? — BAUD - Auto-baud main port

   Yes, go to system boot routine

   No, reset main 9902 and baud the port

3. Test for DSR high on main port

   Not high, print error message

4. Branch to system initialization routine ISYS$ (optional, user supplied)

5. Checksum EPROMs from >F800 to >FFFA — CKSM - Checksum EPROM and ask for devices

   If nonzero sum, print error message

6. If no switches assembled in then

   Ask to select each device present and set bits in DRSEL

(8.2.3 BOOT PROGRAM continued)

7. Output boot menu and get reply If # < 100 then go boot system If # > 100 then go to BOOT utility or user routine	MAIN - Output menu and get reply
8. Auto select disk from disk 0, 4, 8 or 12 based on the switch settings	MAIN04 - Select a boot disk from the switches (get lowest)
9. Save disk # and boot sector # in PARMS	MAIN06 - Boot PDOS system
10. Print 'BOOT..'	
11. Initialize all devices present by calling all INIT DSR entries	
12. Clear sides and density flags and boot the system	
13. Set auto-start and disk # for PDOS	
14. Check auto-boot If auto-booting, just GOOO If not, test for a character	
15. Check for interim character If character, return to MAIN If not, print 'HIT RETURN'	
16. Go to PDOS with a BLWP @>0000 and exit BOOT program	GOOO - Go to PDOS

8.2.3.1 ISYS$ LINKAGE

Sometimes it is desirable to call a routine from the boot program only once. This is done with the System Initialization routine. This routine, if present, is called once via a Branch and Link instruction after the port is bauded and immediately before the EPROM is checksummed. All registers may be used without affecting BOOT operation. Possible uses for the ISYS$ routine include prompting the user for a disk step rate constant, printing a long page of instructions, or performing some other one-time procedure.

The standard 101, 102 and Video Games PDOS EPROMs do not use an ISYS$ routine. The standard STD PDOS EPROM uses ISYS$ to set the stepping rate constant for the FD1793 Floppy controller chip. (See 'ISSTD:SR' for an example of ISYS$.)

To write a routine, simply set the origin at relocatable >0000 and externally define the label ISYS$ equal to the subroutine entry point. Since the address to return to the boot program is passed in R11, the ISYS$ routine is exited with a B *R11, or its equivalent. Any of the BOOT:SR subroutines may be used in the ISYS$ module.

```
        DEF ISYS$
        RORG 0
ISYS$   ...             ;ENTER ROUTINE
        ...
```

8.2.3.2 DSR MODULE LINKAGE

System constants, locations, and subroutine addresses defined by the BOOT:SR program are externally defined for use in the DSRs and the user programs. The list to the right is extracted directly from BOOT:SR. The DEF'ed entries are defined as follows:

TICS2 — This label is the address of the PDOS system 16-bit time counter, It is incremented every time the system clock interrupts (each TIC). Possible DSR uses include waiting for a certain number of seconds (use TPS to calculate one second) and checking elapsed time.

TIME — This label is the address of the PDOS system location of the current task timer. When a task is scheduled, it is loaded with the number of TICs the task is allowed. Each subsequent clock interrupt decrements TIME. When it goes to zero or negative, PDOS swaps to the next task. This counter can be cleared at any time by a DSR to avoid wasting CPU time while waiting for an elapsed time or for a completion signal.

TPS — This label is a system constant equal to the number of TICs, TIME increments, that constitute one second in real time. One possible DSR use is to wait one second, by sampling TICS2 at the beginning, calculating the delta TICs in a loop and comparing the difference to the constant TPS.

D303C — This label is the address of the PDOS system location of a ten word memory block used by the BT303:SR DSR to pass to the 303A the R/W sector command list. These locations are only used in BT303 on the TM990/101MA system so that the paging ER3232 RAM cards can be used with the 303A controller.

```
       DEF TICS2,TIME,TPS
       DEF D303C,L3LOCK,SWLOCK
       DEF DSFLG,DDFLG,MOFLG
*
       DEF BAUD,BWS,IBWS
       DEF MAIN,PARMS,BUFF
*
       DEF BTTO,BBTTO,BTGC
       DEF BTGN,BOUTH,BTPM
       DEF XDITB
*
* USER ROUTINES
*
       REF USER$  ;USER UTILITIES
       REF ISYS$  ;SPECIAL INITIALIZATION
*
****************************************
*      REQUIRED LINK PARAMETER LIST
****************************************
*
*             ;UNIT   TM990   STD95  VG
*
   REF XDLT01   ;0-3    303A    FDC/1  9909
   REF XDLT02   ;4-7    ER3314  SASI   RS232
   REF XDLT03   ;8-11   210     RS232  --
   REF XDLT04   ;12-99  ER3300  --     --
```

(8.2.3.2 DSR MODULE LINKAGE continued)

L3LOCK — This label is the address of the PDOS system location of the level 3 lock. This location locks out all other task's from executing level 3 disk routines. PDOS sets L3LOCK to -1 before it branches to the R/W sector routines. Every DSR MUST clear this lock before exiting back to PDOS, so that others (including himself) can enter again. Non-DMA DSRs should 'ABS @L3LOCK' at the beginning of the routine to allow page swapping.

SWLOCK — This label is the address of the PDOS task lock flag. When non-zero, PDOS does not swap, regardless of how many TICs go by. This is the lock flag that is set and reset by with the XLKT and XULT primitives. One possible DSR use is to lock out the execution of other tasks during time critical device servicing, to avoid data overruns. Note that this lock only inhibits all other tasks from executing, NOT all other code. The clock interrupt routine continues to execute as well as other interrupt processors, such as characters or hardware events, even though SWLOCK is set. For more time critical parts of DRS code, a LIMI 0 must be executed to inhibit even the clock from interrupting. But beware, or the system real time clock can lose TICs.

DSFLG — This label is the address of a four byte disk sides table. Each byte corresponds to one disk: a zero indicates that the disk is single sided and a >FF indicates double sided. The floppy disk DSRs normally set these flags from byte 30 of the header sector, after any successful reading of sector 0. This allows the DSRs to automatically handle both single and double sided media, transparent to the user.

(8.2.3.2 DSR MODULE LINKAGE continued)

DDFLG This label is the address of a four byte disk density table. Each byte corresponds to one disk: a zero indicates that the disk is single density and a >FF indicates double density. The floppy disk DSRs normally set these flags from byte 31 of the header sector, after any successful reading of sector 0. This allows the DSRs to automatically handle both single and double density formatted media, transparent to the user.

MOFLG This label is the address of a four byte motor on table. The use of these locations varies with different DSRs. For BT3300, each byte corresponds to one disk. Whenever a disk is accessed, a byte constant of 10 is loaded, indicating a ten second timeout. When PDOS calls the DOFF entry of the DSR, the logic decrements each positive byte and turns off the corresponding drive motor if it equals zero.

The 12 byte flags described above can be redefined for any purpose by the user, if the devices that use them are not to be linked. See the DSR definition section for a list of where they are used.

BAUD This label is the address of the BOOT routine that auto-bauds the main port and continues on the start the boot program.

BWS This label is the address of the BOOT program's main workspace.

IBWS This label is the address of the BOOT programs's secondary workspace. This is used by BOOT for R/W sector calls, but you can use it for other BLWPs in your auxiliary routines.

(8.2.3.2 DSR MODULE LINKAGE continued)

MAIN — This label is the address of the BOOT program routine that prints the main menu and prompts for user selection. This is the address that a USER$ routine branches to when returning the the boot program.

PARMS — This label is the address of the BOOT program's parameter list. This and the subsequent locations store the parameters entered when the boot program subroutine BTGN is called. See the subroutine definition section for details.

BUFF — This label is the address of the BOOT program's buffer area and points just beyond the secondary workspace.

BTTO — This label is the address of the BOOT program subroutine which outputs the one or two characters in R0. See the subroutine definition section for details.

BBTTO — This label is the address of the BOOT program subroutine which outputs the one or two characters following the call, in *R11. See the subroutine definition section for details.

BTGC — This label is the address of the BOOT program subroutine which gets a character from the main port into the MSB of R0. See the subroutine definition section for details.

BTGN — This label is the address of the BOOT program subroutine which outputs a prompt message and gets either a carriage return or a list of decimal numbers. The numbers are converted to binary, stored in order beginning at location PARMS, and the last one entered is passed back in R1. See the subroutine definition section for details.

(8.2.3.2 DSR MODULE LINKAGE continued)

BOUTH — This label is the address of the BOOT program subroutine which outputs the hexadecimal value of R3 to the main port. See the subroutine definition section for details.

BTPM — This label is the address of the BOOT program subroutine which outputs the message whose address immediately follows the call, in *R11. See the subroutine definition section for details.

XDITB — This label is the address of the BOOT program subroutine which calls the initialize routine of all the installed DSRs. This can be used in format routines and the USER$ program. See the subroutine definition section for details.

Labels defined in BOOT:SR

The following label are externally referenced by BOOT:SR, or REF'd in, and must be externally defined, or DEF'd out, by DSRs and USER$ routines. BOOT:SR assembles default routine and table addresses so that if no subsequent module, either DSR or user routine, externally defines the following labels, then the boot program ignores the call altogether.

USER$ — This label is the entry address of the auxiliary routine that you link into the boot EPROMs. See the USER$ linkage section for more details.

ISYS$ — This label is the entry address of the system initialization routine that you link into the boot EPROMs. See the ISYS$ linkage section for more details.

Disks 0-3

XDLT01 — This label is the address of the DSR link table for device #1. When this device is installed and selected, any PDOS calls for disk numbers 0-3 reference this table. This label must be externally defined, or DEF'd out, by the DSR module for device #1.

(8.2.3.2 DSR MODULE LINKAGE continued)

XDLT02	This label is the address of the DSR link table for device #2. When this device is installed and selected, any PDOS calls for disk numbers 4-7 reference this table. This label must be externally defined, or DEF'd out, by the DSR module for device #2.	**Disks 4-7**
XDLT03	This label is the address of the DSR link table for device #3. When this device is installed and selected, any PDOS calls for disk numbers 8-12 reference this table. This label must be externally defined, or DEF'd out, by the DSR module for device #3.	**Disks 8-11**
XDLT04	This label is the address of the DSR link table for device #4. When this device is installed and selected, any PDOS calls for disk numbers 12-99 reference this table. This label must be externally defined, or DEF'd out, by the DSR module for device #4. The default boot sector index into the DSR link table for these devices is equal to the disk number modulo 4. For example, booting from disk 52 begins at the sector indicated by the first, or disk 0, boot sector entry since 52 is congruent to zero, modulo 4.	**Disks 12-99**

8.2.3.3 MEMTEST

Option 100 of the boot EPROMs selects a memory test routine. An optional second decimal parameter is used to select a memory test range other than from >0000 to >7000. The routine first passes through memory, writing random data. A second pass is then made to verify the data. For each successful memory pass, a period and bell are output to the console. If an error occurs, the address is printed along with the exclusive OR of the data read and the correct data. To exit the memory test, you must restart the boot program with either the RESET or LOAD vector.

```
?100,57312      Tests >0000->E000
?100,53216      Tests >0000->D000
?100,49120      Tests >0000->C000
?100,45024      Tests >0000->B000
?100,40928      Tests >0000->A000
?100,36832      Tests >0000->9000
?100,32736      Tests >0000->8000
```

8.2.3.4 MEMORY INSPECT/CHANGE

Using the memory inspect and change, system memory is 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 is 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 boot program menu. Entire words are altered, so single byte changes are not 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 boot program menu.

(8.2.3.4 MEMORY INSPECT/CHANGE 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 the 102 map registers at >0080 and the boot workspace at either >7000 or >F000 must not be disturbed.

8.2.3.5 OTHER BOOT

A system boot is read into memory from any desired disk number beginning at any logical sector number by using the OTHER BOOT option. Since the default boot sector number is in EPROM, OTHER BOOT is useful when booting from a non-standard disk, or for checking the ability to write data to and read it back from a device. The OTHER BOOT routine reads data from any disk # beginning at any logical sector number into memory from >0000 to >6000. The reply to the prompt is disk #, comma and sector #. As with SYSTEM BOOT, if you hit a character from the console during the boot process, the program booted is not entered. Instead, control returns to the boot program menu and prompt.

```
UNT,SCT=
UNT,SCT=1,10098<CR>
```

8.2.3.6 MAKE BOOT

A system boot is written to any desired disk number beginning at any logical sector number by the MAKE BOOT option. This is useful in backing up the PDOS boot to another disk, or for checking the ability to write data to and read it back from a device. The MAKE BOOT routine writes memory from >0000 to >6000 to any disk # beginning at any logical sector number. The reply to the prompt is disk #, comma and sector #. Any errors are reported and control returns the the boot program menu and prompt.

```
UNT,SCT=
UNT,SCT=3,1846<CR>
```

8.2.3.7 USER$ - AUX UTILITIES

If there is room left in the boot EPROMs, you can add auxiliary routines. These modules are linked with BOOT:SR and the other DSR modules. Care must be taken that there is enough room for all the routines. Simply set the origin at relocatable >0000 and externally define the label USER$ equal to the routine entry point. Then the boot menu selection 104 branches to your routine. Upon transfer to the USER$ routine:

R13 points to the character output routine ,BBTTO
R14 points to the get hex routine, BTGH
R15 points to the output hex routine, BOUTH

All boot subroutines may be used. Multiple auxiliary routines are added by making another menu as the main routine. To exit from the routine and return to the main boot menu, you must branch to the external reference MAIN.

```
        DEF USER$
        RORG 0
USER$   ...             ;ENTER ROUTINE
        ...

        REF MAIN
        ...
        B @MAIN         ;EXIT
```

8.2.4 SYSTEM BOOT

The boot program needs to two which devices are installed in the system and whether or not to auto-boot. The first is obtained from either the sense switches or from responses from the operator. The second comes from either the sense switch or the BTFLG:SR flag, FLGAB. If the auto-boot feature is taken, then PDOS performs an auto-start, too.

BOOT:SR uses system memory locations in PDOS to pass this information to PDOS initialization. First, the memory byte at location >0070 (which is in the XOP 12 vector location), is the auto-start flag for PDOS. If it is zero, PDOS auto-bauds the port and prompts for the date and time as usual. If it is non-zero, PDOS does not auto baud, but executes the file SY$STRT instead. If either the auto-boot feature is selected or byte >0070 is non-zero on the boot disk, then the auto-start is initiated.

Second, the memory byte at location >0071 (also in the XOP 12 vector), is loaded byte the boot program with the disk number from which the system was booted. PDOS loads this value into the default system disk number for task 0 at system initialization.

If no sense switches are used, the boot program loads the memory word at location >2FFE with a bit encoded value which tells the R/W sector handler in BOOT:SR which devices are installed. The least significant bit of the memory word corresponds to device #1 and the bit is set if the device is installed.

When booting the system, the boot program calls the appropriate DSR read sector routine with BLWP instruction, just like PDOS does. However, the secondary boot workspace is used.

8.2.4.1 DEVICE SELECTION

Device selection information is given to the boot program by either sense switches or through an operator prompt sequence. Only the 101 and 102 PDOS systems may have sense switches, but all PDOS systems may be configured to use the operator prompts.

(8.2.4.1 DEVICE SELECTION continued)

SENSE SWITCHES

Some PDOS systems have sense switches available to read under program control. These are the TM990/101MA CPU and the TM990/102 CPU used in conjunction with a TM990/307 I/O card. The standard 101 PDOS EPROMs use switches to select devices and auto-booting. The standard 102 EPROMs and all other PDOS systems do not assume that switches are present, but rather they use the operator prompt sequence for determining devices present.

The BOOT:SR program defines the function of the TM990/101MA switches as follows:

```
FLG101  EQU 1           ;SELECT 101
FLGSW   EQU 1           ;USE SWITCHES
```

S1 = ON = Auto-boot & execute 'SY$STRT'
S2 = ON = 303A controller (UNITS 0-3)
S3 = ON = 3314 Winchester controller (UNITS 4-7)
S4 = ON = 210-3 bubble card (UNITS 8-11)
S5 = ON = 3300 floppy controller (UNITS 12-99)

These switch definitions are included in BOOT only if FLG101 equals 1 and FLGSW equals 1.

To assemble a 102 EPROM with 307 switches, set FLG102 equal to 1 and FLGSW equal to 1. The BOOT:SR program then defines the function of the TM990/307 switch pack S8 as follows:

```
FLG102  EQU 1           ;SELECT 102 CPU
FLGSW   EQU 1           ;ASSUME 307 IN
```

S8 = OFF = Auto-boot & execute 'SY$STRT'
S7 = OFF = 303A controller (UNITS 0-3)
S6 = OFF = 3314 Winchester controller (UNITS 4-7)
S5 = OFF = 210 bubble card (UNITS 8-11)
S4 = OFF = controller #4 (UNITS 12-99)

The special assembler option, #, is used to either assemble the switch logic (#=1) or the select logic (#=0).

OPERATOR PROMPTS

When no switches are available, the BOOT program prompts to select those devices whose DSR's were assembled into the EPROM are currently present in the system. The message from each DSR is output, preceded by the word 'SELECT' and followed with a question mark. You simply enter a 'Y', if the device is in the system, or a carriage return, if not. This sets bits in the location DRSEL. DRSEL is used by the boot to determine legal disk numbers.

```
SELECT 303A ?
```

(8.2.4.1 DEVICE SELECTION continued)

Only one device DSR is called for Disk Off service. This device is indicated to BOOT:SR by the flag NOFF. If set to zero, no motor off DSR entries are called by the EPROM. If NOFF is set equal to 1, then the XDLT01 device motor off entry is called once a second (if it is present).

```
NOFF    EQU 0           ;NO MOTOR OFF DEVICE

NOFF    EQU 4           ;UNITS 12-99 REQUIRE SERVICE
```

8.2.4.2 AUTO-BOOT

The PDOS boot EPROMs have the facility to automatically boot PDOS into RAM and set the auto-start flag at memory byte address >0070. On a TM990/101MA system, this option is selected by switch #1 on the CPU card. Other systems require external switches or hard coded auto-boot. If the flag FLGAB in the BTFLG:SR file is set to a 1, then the resulting boot EPROMs auto boot even without switches.

```
Switch 1 ON = Auto boot

FLGAB   EQU 1           ;AUTO-BOOT W/O SWITCHES
```

8.2.4.3 AUTO-START

If the auto-start flag (byte >0070) is non-zero, then PDOS automatically executes the file named 'SY$STRT' on the system disk. Care must be taken that a baud port command (BP) is executed under control of the 'SY$STRT' file, since the system console port is not auto-bauded.

The file type of 'SY$STRT' indicates how the file is to be executed. Normally, it is a procedure file (typed AC) with the first command being a BAUD PORT (BP) for the console port. Other commands might include configuring other user tasks and the starting of a turn-key application program. The name of the auto-start file is changed using the BFIX utility. If byte >70 on the boot disk is non-zero, auto-start is entered regardless of the switch setting of the value of FLGAB in the ROMs.

```
.SA SY$STRT,AC
.SF SY$STRT
BP 1,19200
BP 2,9600
SY 1
LV 10
MENU
RC
.
```

Byte location >0071 is loaded with the boot disk number after the system is booted and just before a 'BLWP @>0000' is executed. PDOS loads the default system disk number from this location. Thus, the system comes up using the same disk from which it was booted.

```
>0070 = Auto-start flag
>0071 = Initial default disk #
```

8.2.5 BOOT SUBROUTINES

BOOT:SR externally defines some of its utility subroutine addresses so that they are available for user routines. The following summary defines the function and register usage of each. These subroutines may be used in USER$ and ISYS$ modules, but usually not by DSRs. However, for debug purposes they might even come in handy for outputting characters on retries or intermediate error numbers.

8.2.5.1 BTTO - OUT RO

Function: This routine outputs to the main port, the character(s) in RO.

Call sequence:
```
LI RO,'OK'
BL @BTTO
```

Registers: Destroys R9 & R12.

8.2.5.2 BBTTO - OUT *R11

Function: This routine outputs to the main port, the character(s) following the call.

Call sequence:
```
BL @BBTTO
  DATA 'OK'
```

Registers: Destroys R9 & R12.

8.2.5.3 BTGC - GET CHARACTER AND ECHO

Function: This routine gets a character from the main port, stores it in the left byte of RO, and echoes it if it is printable.

Call sequence:
```
BL @BTGC
```

Registers: Destroys R9 & R12.

(8.2.5 BOOT SUBROUTINES continued)

8.2.5.4 BTGN - GET NUMBER

Function: This routine outputs a prompt message, gets a series of decimal numbers separated by commas, and returns the last number entered in R1. All the numbers are stored in order beginning at location @PARMS, with the next un-entered parameter zeroed.

Call sequence:
```
BL @BTGN
  DATA PROMPT           ;MESSAGE TERMINATED WITH BYTE 0
  DATA PROBLEM ADDRESS  ;ROUTINE TO HANDLE ILLEGAL CHAR
  <CR> RETURN           ;CR ONLY
NORMAL RETURN
```

Registers: Destroys R0-R3 & R9-R12.

8.2.5.5 BOUTH - OUT HEX R3

Function: Outputs to the main port the hexadecimal value of R3. Only 4 characters are output.

Call sequence:
```
BL @BOUTH
```

Registers: Destroys R0, R2-R5, R9 & R12.

8.2.5.6 BTPM - PRINT MESSAGE

Function: Output to the main port the message whose address immediately follows the call.

Call sequence:
```
BL @BTPM
  DATA MES01
```

Registers: Destroys R0, R1, R5, R9 & R12.

8.2.5.7 XDITB - INIT DEVICES

Function: Initialize all devices present for format or INIT utilities.

Call sequence:
```
BL @XDITB
```

Registers: Depends on the DSR's called.

8.3 DSR MODULES

All PDOS Device Service Routines come from source files whose names begin with the letters 'BT' (for Boot file), followed by up to 6 letters indicating the particular device and having an 'SR' extension, meaning that it is an assembly source. BT303:SR is the TM990/303A floppy disk controller DSR and BT210:SR contains the DSR for the TM990/210 bubble memory board. You can name your files differently if you desire.

The DSR program begins at relocatable origin >0000. The LINKer assigns the final absolute address and resolves all references. In general, each DSR is called for only 4 disk numbers. For example, the standard Boot EPROMs call BT303, the DSR for the 303A, only if PDOS is accessing disk numbers 0 through 3. This is only a function of BOOT:SR, and you can vary this if needed.

At the beginning of the program is the LINKAGE TABLE. This table contains data needed by the BOOT module for proper DSR operation. The table consists of four (4) jump instructions, four DATA constants indicating the default boot sectors of the 4 disks, and a TEXT string (terminated with a BYTE 0) which is printed by BOOT for the device selection prompt.

The four jump entries transfer to the four required DSR routines. They are: 1) device controller initialization; 2) a logical sector read; 3) a logical sector write; and 4) motor off functions.

The DSR initialization routine is called via a Branch and Link (BL) instruction before booting and during PDOS startup. Registers R0 through R13 may be used and return address is passed in R11. Do not use R14 and R15. Typical functions performed include resetting the controller, restoring all disk drives, setting drive dependent parameters, or sending some other initial commands. If no controller initialization is needed for the device, an RT return can replace the JMP instruction in the LINKAGE TABLE.

The logical sector read and write routines are called via a Branch and Load Workspace Pointer (BLWP) instruction. Parameters are passed to them in registers R0, R1 and R2 of the calling workspace and, therefore, must be fetched using R13.

LINK FORMAT:

```
XDLTxx  JMP DINIT       ;INITIALIZE DEVICE
        JMP DREAD       ;READ LOGICAL SECTOR
        JMP DWRIT       ;WRITE LOGICAL SECTOR
        JMP DSCOF       ;DRIVE OFF (1 SEC)
        DATA BS0        ;BOOT SECTOR 0
        DATA BS1        ;BOOT SECTOR 1
        DATA BS2        ;BOOT SECTOR 2
        DATA BS3        ;BOOT SECTOR 3
        TEXT '...',0    ;DRIVER NAME


DINIT   ....            ;INITIALIZE DEVICE
        ....            ;(USE ONLY R0-R13)
        RT


DREAD   ....            ;READ LOGICAL SECTOR
DWRIT   ....            ;WRITE LOGICAL SECTOR
        CLR @L3LOCK     ;CLEAR LEVEL 3 LOCK
        INCT R14        ;NORMAL RETURN
        RTWP
```

(8.3 DSR MODULES continued)

The logical disk number is located in *R13, the logical sector number to read/write is in @2(R13), and the logical buffer address is in @4(R13). These routines exit with a Return Workspace (RTWP) instruction, so registers R13, R14 and R15 must be preserved. If no error is encountered, the Read/Write routines skip over the return address by executing a 'INCT R14' instruction before returning. If an error is encountered, the DSR loads the error number into the calling workspaces R0, by moving *R13, and does not increment the return address. In either case the level 3 lock flag must be cleared before exiting these routines.

The Motor Off routine is called by PDOS once each second to service controller devices needing constant attention. This routine is called via a Branch and Link (BL) instruction, may only use registers R0, R1, R2 and R12, and exit with a B *R11 instruction. If no attention is required (e.g. 303A board), a RT instruction can replace the jump in the LINKAGE TABLE. This routine could be used with a 'dumb' floppy controller to deselect drives after a certain time or turn off the motors of 5" drives. Certain locations in PDOS, namely MOFLG, are available to keep counters for these types of functions. Care must be taken that more than one DSR doesn't use the same location as a counter or flag.

```
DSCOF   ....            ;DRIVE OFF ROUTINES
        ....            ;(USE ONLY R1-R2,R12)
        RT
```

Note: NO PDOS calls are legal in a DSR. It does help system response to swap to other tasks while waiting for certain timing loops or particular device events. This can be done by clearing the task timer, TIME, inside the loop. (See the subroutine SEEK in BT3300:SR module for an example of how to pause exactly one second.)

NO PDOS calls within a DSR

Swap with CLR @TIME

Sometimes, devices require fast, sure response from the host system. In place of the task lock/unlock primitives, you simply 'set to ones' the swap lock location, SWLOCK, to inhibit PDOS from taking control from you. When you are through, be sure to release the system by unlocking your process, CLR @SWLOCK.

Lock task: SETO @SWLOCK

Unlock task: CLR @SWLOCK

This still allows the execution of interrupt service routines, such as the system clock, 9902 characters, and hardware events. If the timing is too critical for event these processes to be occurring, then the most drastic action, disabling all interrupts with a LIMI 0, must be used. Be careful that you exit from the critical code: 1) gracefully, restoring the mask to the same level it was before you zeroed it, and 2) quickly, so that the PDOS system clock doesn't lose any TICs over your indiscretion. See the BT303:SR module for a trick to restore the interrupt mask to its original level.

8.4 GENERATING BOOT EPROMS

The boot EPROMs for the standard PDOS systems are generated with procedure files. The file named DOBOOT:101 generates the object file from which the TM990/101MA ROMs are burned, the file named DOBOOT:102 generates the object file from which the TM990/102 ROMs are burned, and so on. You need to set the flags in BTFLG:SR before generating any boot program.

As an example of the method used in generating boot ROMs, a listing of the file DOBOOT:101 follows:

```
SF BTFLG:SR                    Show the boot flags
ASM BOOT:SR,#BOOT;6            Assemble BOOT and the DSR modules
ASM BT303:SR,#BT303;6          using those flags
ASM BT3314:SR,#BT3314;6
ASM BOOTE:SR,#BOOTE;6
LINK                           Invoke the Linker
0,BOOT                         Output to the BOOT file
8,>F000                        Set the buffer base to >F000
1,BOOT                         Link in BOOT and the DSR's
1,BT303
1,BT3314
8,>FFFC                        Add the LOAD vector for 101's
1,BOOTE
2                              List problems
3
4,TEMP                         Output the link map for examination
6                              Set entry tag and quit
7                              Exit LINK
LOGO                           Invoke the loader to set the checksum
2,>F000                        Set the buffer base
1,BOOT                         Get the linker output
3,>F800,>FFFA,>F812            Set the checksum
4,>F800,>FFFF,BOOT             Output for burning
6                              Exit LOGO
RC                             End of chain file
```

Note the the LOGO utility is used only to set the checksum in the EPROMs. The LOGO command line that sets the checksum (3,>F800,>FFFA,>F812) is common to all PDOS systems, since BOOT:SR uses the same code in all systems to checksum the ROMs.

Checksum from >F800 to >FFFF and place value in location >F812

Care must be taken when generating ROMs that the modules do not overlap. The Linker doesn't check to see if a location is addressed more that once, so you must examine the link map file closely, checking the each module's beginning and ending addresses.

(8.4 GENERATING BOOT EPROMS continued)

The following is the console output resulting from executing the chain file, DOBOOT:101, listed above. Note that only addresses above >F800 are used since 101 PDOS only has two 2708 EPROMs, or 2k bytes of memory, for the boot. Other utilities may be added to the 101 boot program, but this would require adding two more 2708's to the 101 CPU card.

```
.DOBOOT:101                                          Start the chain file
.SF BTFLG:SR                                         Show the flags for verification
*       BTFLG:SR        09/17/82
*************************************************
*       SYSTEM CONFIGURATION FLAGS
*************************************************
*
FLG101  EQU 1           ;TM990/101M                  Set for 101 system
FLG102  EQU 0           ;TM990/102
FLG95S  EQU 0           ;STD TMS9995
FLG95V  EQU 0           ;VIDEO GAMES TMS9995
*
FLGROM  EQU 0           ;0=RAM, 1=EPROM
FLGAB   EQU 0           ;0=AUTO-BAUD, 1=AUTO-BOOT
FLGSW   EQU 1           ;0=NO SWITCHES, 1=SWITCHES   Set for switches
LDV1    EQU 1           ;LOGICAL DEVICE #1 (XDLT01)  Set for 303A
LDV2    EQU 1           ;LOGICAL DEVICE #2 (XDLT02)  Set for 3314
LDV3    EQU 0           ;LOGICAL DEVICE #3 (XDLT03)
LDV4    EQU 0           ;LOGICAL DEVICE #4 (XDLT04)
*
NOFF    EQU 0           ;DISK OFF DEVICE #           No disk off DSRs
*
FLG95   EQU FLG95S!FLG95V
DVSEL   EQU LDV4*2+LDV3*2+LDV2*2+LDV1
.ASM BOOT:SR,#BOOT;6                                 Assemble the general boot program
ASM R2.4
SRCE=BOOT:SR
OBJ=#BOOT;6
LIST=
ERR=
XREF=

END OF PASS 1
0 DIAGNOSTICS
END OF PASS 2
0 DIAGNOSTICS
```

(8.4 GENERATING BOOT EPROMS continued)

```
.ASM BT303:SR,#BT303;6                          Assemble 303A DSR
ASM R2.4
SRCE=BT303:SR
OBJ=#BT303;6
LIST=
ERR=
XREF=

END OF PASS 1
0 DIAGNOSTICS
END OF PASS 2
0 DIAGNOSTICS
.ASM BT3314:SR,#BT3314;6                        Assemble ER3314 DSR
ASM R2.4
SRCE=BT3314:SR
OBJ=#BT3314;6
LIST=
ERR=
XREF=

END OF PASS 1
0 DIAGNOSTICS
END OF PASS 2
0 DIAGNOSTICS
.ASM BOOTE:SR,#BOOTE;6                          Assemble 101MA LOAD vector module
ASM R2.4
SRCE=BOOTE:SR
OBJ=#BOOTE;6
LIST=
ERR=
XREF=

END OF PASS 1
0 DIAGNOSTICS
END OF PASS 2
0 DIAGNOSTICS
```

(8.4 GENERATING BOOT EPROMS continued)

```
.LINK                                                    Invoke the Linker
LINKER R2.4
*0,BOOT
*8,>F000
WAS >0000
*1,BOOT
*1,BT303
*1,BT3314
*8,>FFFC
WAS >FFDC                                                Note: didn't overrun EPROM
*1,BOOTE
*2
UNDEFINED DEF ENTRIES:
ISYS$  >0000  USER$  >0000  XDLT03 >0000  XDLT04 >0000
*3
MULTIPLY DEFINED DEF ENTRIES: NONE
*4,TEMP
*6
START TAG = >0000
*7                                                       Exit LINK
.LOGO                                                    Invoke the loader for checksumming
LOGO R2.4
*2,>F000
*1,BOOT
LOADING.....
IDT='BOOT2.4 '                                           Note: IDT order
IDT='B3032.4 '
IDT='B3142.4 '
IDT='BOOTEV  '
ENTRY ADDRESS=>0000
*3,>F800,>FFFA,>F812                                     Set checksum in object
*4,>F800,>FFFF,BOOT
*6                                                       Exit LOGO
.RC                                                      End of chain file
```

(8.4 GENERATING BOOT EPROMS continued)

The link map generated above is listed below. Note that the last DSR module, BT3314, is located up to address >FFDC, indicating that only 30 bytes are available in the EPROMs. Also, the only undefined external references are ISYS$ (the optional system initialize routine), USER$ (the optional auxiliary routine), and the two undefined DRS's, XDLT03 and XDLT04.

```
.SF TEMP

LINK MAP FILE
TIME=11:49:45
DATE=09/17/82
FILE=BOOT
FILE  NAME:EXT   IDT         ENTRY  [--PSEG---]  [--DSEG---]  DEF  REF

   1  BOOT      'BOOT2.4 '   >F800  >F000 >FCD4  >0000 >0000   20    6
   2  BT303     'B3032.4 '   >FCD4  >FCD4 >FE86  >0000 >0000    1   12
   3  BT3314    'B3142.4 '   >FE86  >FE86 >FFDC  >0000 >0000    1    2
   4  BOOTE     'BOOTEV  '   >FFFC  >FFFC >0000  >0000 >0000    0    2

DEF      F/T  VALUE   REFERENCES

BAUD     1 P  >F89C   #4 P >FFFE
BBTTO    1 P  >FBDC
BOUTH    1 P  >FB4A
BTGC     1 P  >FBAA
BTGN     1 P  >FB62
BTPM     1 P  >FB3C
BTTO     1 P  >FB8E
BUFF     1 A  >7060
BWS      1 A  >7000   #4 P >FFFC
D303C    1 A  >2FBC   #2 P >FE14   #2 P >FE22   #2 P >FE2A   #2 P >FE36        DDFLG    1 A  >2FD4
DSFLG    1 A  >2FD0   #2 P >FD32   #2 P >FD78
ISYS$      U  >0000   #1 P >FBFC
L3LOCK   1 A  >2FE8   #2 P >FD62   #3 P >FF8C
MAIN     1 P  >F8FE
MOFLG    1 A  >2FD8
PARMS    1 A  >7024
SWLOCK   1 A  >2FEA
TICS2    1 A  >2F88   #2 P >FDFC   #2 P >FE02
TIME     1 A  >2FEC   #2 P >FE42   #2 P >FE70   #3 P >FFCC
TPS      1 A  >007D   #2 P >FE08
USER$      U  >0000   #1 P >FBF8
XDLT01   2 P  >FCD4   #1 P >FBE8
XDLT02   3 P  >FE86   #1 P >FBEA
XDLT03     U  >0000   #1 P >FBEC
XDLT04     U  >0000   #1 P >FBEE
```

(8.4 GENERATING BOOT EPROMS continued)

Even though burning the two TMS 2708 EPROMs might be accomplished in a chain file, do this interactively. Either the BURNP or the BURN302 utility accepts the output from LOGO for burning the EPROMs. BURNP is an RS232 type burn program where the object is passed to a standalone burner over an RS232 serial link. The actual burning and verification of the EPROMs are done external to PDOS. Therefore, we use the BURN302 utility for this example, so that the loading, burning, and verification can be demonstrated.

```
.BURN302
BURN302 R2.4
*NOTE: ALL NUMBERS ARE HEX.
*
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
*B,F000
HIGHEST PC=0000
BUFFER LIMITS ARE F000 TO FFFF
*1,TEMP
ENTRY ADDRESS=0000
*C,F800,FFFF
CHECKSUM=0000
*A
*P,F800,FFFF,L
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........
VERIFYING
*VF800,FFFF,L
*A
*PF800,FFFF,R
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........+
.........
VERIFYING
*VF800,FFFF,R
*E
A_
```

8.5 DSR DEFINITIONS

The following is a summary of all current Device Service Routines in the PDOS library. The file names, addressing, and format information is included in an abbreviated form. Any current restrictions are also listed. Use this information as a reference only. See the sources for complete information. The definition of the headings precedes the actual module information:

```
        Name: BT<device>:SR
      Device: Device name and description
     Address: Memory mapped and CRU mapped addresses
     Formats: Media formats supported, formatting utility name
        Boot: Location and format of boot sectors
Boot Sectors: General boot sector number
Memory Usage: Registers and/or system memory locations needed
        Size: Module size in decimal bytes [in hexadecimal]
  Device DEF: Device number and DEF'd out label name
Restrictions: Other information about the DSR
      Errors: Error numbers generated by the DSR and their meanings
```

8.5.1 BT210:SR

```
        Name: BT210:SR
      Device: TM990/210-3 Bubble memory module
     Address: Memory mapped at >E100, >E120, >E140, and >E160
     Formats: N/A, no formatting necessary
        Boot: N/A
Boot Sectors: All at sector 156
Memory Usage: Registers
        Size: 272 bytes [>110]
  Device DEF: #3=XDLT03
Restrictions: Bubble boards must be -3 boards (6 bubbles)
      Errors: 101=Sector too large
              102=Controller timeout
```

(8.5 DSR DEFINITIONS continued)

8.5.2 BT232:SR

```
        Name: BT232:SR
      Device: RS232 communication link
     Address: CRU base address at >0180 or >0040 (VG)
     Formats: N/A, no formatting necessary
        Boot: N/A
Boot Sectors: All at sector 1846
Memory Usage: Registers
        Size: 326 bytes [>146]
  Device DEF: #3=XDLT03
Restrictions: Slave system must be executing RS232 as a
              background task.  Communication is at 9600 baud.
      Errors: 101=Invalid response
              102=Line timeout
              103=No acknowledge
```

8.5.3 BT303:SR

```
        Name: BT303:SR
      Device: TM990/303A floppy controller
     Address: CRU base address at >0200
              DMA sector transfers.
     Formats: Single and double sided, double density 8" floppy
              Use the FRMT303 utility
        Boot: Single sided, double density, tracks 72-75
Boot Sectors: All at sector 1846
Memory Usage: Registers
              >2FBC->2FCF Command table
              >2FD0->2FD3 Double sided flags (DSFLG)
        Size: 434 bytes [>1B2]  (102 CPU is larger)
  Device DEF: #1=XDLT01
Restrictions: Uses high extended address lines
      Errors: 102=Controller timeout
              109=Mapper boundary violation
              110=Sides not disk compatible
              others are R0 and R1 words combined
```

(8.5 DSR DEFINITIONS continued)

8.5.4 BT3300:SR

```
        Name: BT3300:SR
      Device: ER3300A floppy controller
     Address: Memory mapped at >E000->E01F
              CRU mapped at >0400->047F
     Formats: Single sided, single and double density 5" floppy
              Use the FRMT93 Utility
        Boot: Single sided, single density, tracks 30-39
Boot Sectors: All at sector 300
Memory Usage: Registers
              >2FD4->2FD7 Double density flags (DDFLG)
              >2FD8->2FDB Motor off flags (MOFLG)
        Size: 492 bytes [>1EC]
  Device DEF: #4=XDLT04
Restrictions: Don't work PAUL.
      Errors: 101=Sector too large
              102=Not ready/controller timeout
              103=Write protected
              104=Write fault
              105=RNF/Seek error
              106=CRC wrong
```

8.5.5 BT3314:SR

```
        Name: BT3314:SR
      Device: ER3314 SASI interface
     Address: Memory mapped at >E020->E03F
              CRU mapped at >0480->04FE
     Formats: Peripheral defined, use the FRMTW utility
        Boot: N/A
Boot Sectors: 0 = 16288
              1 = 16288
              2 = 1846
              3 = 3856
Memory Usage: Registers
        Size: 342 bytes [>156]
  Device DEF: #2=XDLT02
Restrictions: Must be assembled for enabled or disabled
              interrupts during data transfers.  Disk numbers
              0 and 1 select logical device 0.  Disk number 2
              selects logical device 2 as a SA800 type device.
              Disk number 3 selects logical device 3 as a
              SA850 type device.
      Errors: 102=Controller timeout
              200+SASI Error code
```

(8.5 DSR DEFINITIONS continued)

8.5.6 BT95VG:SR

```
        Name: BT95VG:SR
      Device: TMS9909 based MOED 1095 single board computer
              Video Games TMS9995 based SBC
     Address: Memory mapped at >F860->F89F
              CRU mapped at >0140->015E
     Formats: Single and double sided, single density 5" floppy
              Use the FRMTVG utility
        Boot: Single sided, single density, tracks 30-39
Boot Sectors: All at sector 300
Memory Usage: Registers
              >2FD0->2FD3 Double sided flags (DSFLG)
              >2FD8->2FDB Motor off flags (MOFLG)
        Size: 414 bytes [>19E]
  Device DEF: #1=XDLT01
Restrictions: 9909 has problems
      Errors: 300-303=Drive not ready after 1.25 (RESTORE)
              304-311=Recalibrate drive errors
              312-315=TRK 00 .NE. PTRACK (0-3)
              316-319=Drive not ready after 1.25 (FORMAT)
              320-323=Rates not defined
                  329=Write protect on FORMAT
              332-335=Drive not ready after 1.25 (READ)
              336-339=Rated not defined
                  340=Hard sector not found
                  341=ID sync bytes not found
                  342=ID address mark not found
                  343=ID CRC error
                  344=ID bytes not found
                  345=Data sync or AM not found
                  346=Data o/flow error
                  347=Data CRC error
             348-351=Drive not ready after 1.25 (WRITE)
             352-355=Rates not defined
                  356=Hard sector not found
                  357=ID sync bytes not found
                  358=ID address mark not found
                  359=ID CRC error
                  360=ID bytes not found
                  361=Diskette write protected
                  362=Data u/flow error
                  363=Data u/flow error (FORMAT TRACK)
```

(8.5 DSR DEFINITIONS continued)

8.5.7 BTFDC1:SR

```
        Name: BTFDC1:SR
      Device: G W Three FDC-1 STD floppy controller
     Address: I/O mapped at >EFC0->EFCF
     Formats: Single and Double sided, Single and Double density,
              5" or 8" floppies
              Use the FRMTFDC1 utility
        Boot: Single sided, single density 5"
              Single sided, double density 8"
Boot Sectors: All at sector 300 for 5"
              All at sector 1846 for 8"
Memory Usage: Registers
              >2FD0->2FD3 Double sided flags (DSFLG)
              >2FD4->2FD7 Double density flags (DDFLG)
              >2FD8->2FDB Motor off flags (MOFLG)
              >F020->F029 On chip RAM stuff
        Size: 678 bytes [>2A6]
  Device DEF: #1=XDLT01
Restrictions: Mixing 5" & 8" drives not supported, though
              possible
      Errors: 101=Sector # too large
              102=Not ready
              103=Write protected
              104=Write fault
              105=RNF/Seek error
              106=CRC wrong
              107=Lost data
              199=Controller timeout
```

(8.5 DSR DEFINITIONS continued)

8.5.8 BTWINC:SR

```
        Name: BTWINC:SR
      Device: Micro/Sys host adaptor for STD bus
     Address: I/O mapped at >EF90->EF93
     Formats: Peripheral defined, use the FRMTWS utility
        Boot: N/A
Boot Sectors: 0 = 16288
              1 = 16288
              2 = 1846
              3 = 3856
Memory Usage: Registers
        Size: 340 bytes [>15A]
  Device DEF: #2=XDLT02
Restrictions: Non-DMA device. Non-14030 (256 us) support.  Disk numbers
              0 and 1 select logical device 0.  Disk number 2
              selects logical device 2 as a SA800 type device.
              Disk number 3 selects logical device 3 as a
              SA850 type device.
      Errors: 102=Controller timeout
              200+SASI Error code
```

8.6 AN EXAMPLE

The following listing is a skeleton of a generalized Device Service Routine. To create a new DSR, use this framework and add the actual code needed by the device. You cannot use error numbers 0 through 99. These are reserved for PDOS and BASIC.

```
*       BT<name>:SR     <date>
*****************************************************
*                                                   *
*       <device definition and summary>             *
*                                                   *
*       XDLT<#> where <#> = 01, 02, 03, 04          *
*                                                   *
*****************************************************
*
        TITL ' <DSR title> '
        IDT '<DSR idt>'
*
*****************************************************
*       LINKAGE TO BOOT:SR
*****************************************************
*
        DEF XDLT<#>     ;DEVICE LINK TABLE
*
        REF TIME        ;TASK TIMER
        REF DSFLG       ;DOUBLE SIDED FLAG
        REF DDFLG       ;DOUBLE DENSITY FLAG
        REF L3LOCK      ;LEVEL 3 LOCK
*
        COPY BTFLG:SR   ;GET FLAGS
*
*****************************************************
*       <device> CONTROLLER CONFIGURATION
*****************************************************
*
BPS     EQU 256         ;256 BYTES/SECTOR

        <device parameters and equates>
        <e.g. CRU bases, step rates, biases>
```

(AN EXAMPLE continued)

```
*************************************************
*       LINKAGE TABLE
*************************************************
*
        RORG 0
XDLT(#) JMP XINIT               ;INITIALIZE DRIVE ENTRY
        JMP XREAD               ;READ SECTOR ENTRY
        JMP XWRIT               ;WRITE SECTOR ENTRY
        JMP XDOFF               ;DISK OFF ENTRY
        DATA (boot sector #0)   ;DISK #0 BOOT SECTOR
        DATA (boot sector #1)   ;DISK #1  "     "
        DATA (boot sector #2)   ;DISK #2  "     "
        DATA (boot sector #3)   ;DISK #3  "     "
        IFN FLGSW,XDLTE
        TEXT '(device name)'    ;DRIVE NAME
        BYTE 0
*
XDLTE   EVEN
*
*************************************************
*       (device) INITIALIZATION ENTRY
*************************************************
*
XINIT   MOV R11,R13     ;SAVE RETURN

        (Can use all registers except R14 & R15)

        B *R13          ;RETURN
*
*************************************************
*       (device) DISK OFF ENTRY
*************************************************
*
XDOFF   EQU $           ;DISK OFF

        (Use only registers R0-R2, and R12)

        B *R11          ;RETURN
*
*************************************************
*       (device) READ SECTOR ENTRY
*************************************************
*
XREAD   EQU $           ;READ SECTOR

        (get read parameters)

        JMP XCRW        ;GOTO COMMON R/W ROUTINE
```

(AN EXAMPLE continued)

```
*************************************************
*       <device> WRITE SECTOR ENTRY
*************************************************
*
XWRIT   EQU $           ;WRITE SECTOR

        <get write parameters>

*
*       COMMON R/W ROUTINE
*
XCRW    MOV *R13,R0     ;GET DISK #
        ANDI R0,>0003   ;MOD(UNIT,4)
        MOV @2(13),R1   ;GET SECTOR NUMBER
        CI R1,<size>    ;SECTOR OK?
          JH ER101      ;N, ERROR
        MOV @4(13),R2   ;Y, GET BUFFER ADDRESS

        <read/write routines>
        <R0 = disk #>
        <R1 = sector #>
        <R2 = buffer address>

*************************************************
*       OPERATION SUCCESSFUL EXIT
*
XRWOK   INCT R14        ;SUCCESSFUL OPERATION
*
*************************************************
*       COMMON RETURN
*
XRWRT   CLR @L3LOCK     ;CLEAR LOCK FLAG
        RTWP
*
ER101   LI R0,101       ;SECTOR TOO LARGE
*
*************************************************
*       ERROR PROCESSING
*
ERROR   MOV R0,*R13     ;RETURN ERROR #
        JMP XRWRT
        END XDLT<#>
```