PDOS®

Realtime Operating System

Reference Manual

Printed in the United States of America.
Product number ER3510/M (for PDOS revision 3.2)
January 1, 1987

CHAPTER 1 -- INTRODUCTION

CHAPTER 2 -- PDOS SYSTEM OVERVIEW

(TABLE OF CONTENTS continued)

CHAPTER 3 -- PDOS MONITOR COMMANDS

(TABLE OF CONTENTS continued)

CHAPTER 4 -- PDOS ASSEMBLY PRIMITIVES

(TABLE OF CONTENTS continued)

(TABLE OF CONTENTS continued)

(TABLE OF CONTENTS continued)

CHAPTER 5 -- PDOS SCREEN EDITOR

(TABLE OF CONTENTS continued)

CHAPTER 6 -- ASSEMBLE AND LINK

(TABLE OF CONTENTS continued)

(TABLE OF CONTENTS continued)

(TABLE OF CONTENTS continued)

CHAPTER 7 -- PDOS UTILITIES

USER UTILITIES

(TABLE OF CONTENTS continued)

SYSTEM FACILITIES

SYSTEM-SPECIFIC UTILITIES

CHAPTER 8 -- BIOS, UARTs, DISK DSRs

APPENDIX A -- PDOS ERROR DEFINITIONS

(TABLE OF CONTENTS continued)

CHAPTER 1

INTRODUCTION

PDOS is a powerful realtime, multi-tasking operating system developed by Eyring Research Institute, Inc., for the Motorola 68000 microprocessor family. Chapter 1 is intended to give you a flavor of the operating system environment.

1.1 HOW TO USE THIS MANUAL

This manual is designed to be a comprehensive reference manual to the PDOS operating system. It covers all monitor commands, assembly primitives, and utilities. Examples and a full demonstration session are also provided. This manual is not a beginner's guide or a tutorial. Other manuals such as "Getting Started With PDOS" or "Installation and Systems Management" as well as PDOS training courses will help you as a beginning PDOS user.

This manual

Each chapter is marked by a tab, with a table of contents for that chapter located at the tab. You may also find, at some tabs, appropriate summaries of the material in the chapter. These pages are supplementary to the text itself. Since they are not numbered, you may remove them from the binder and use them for reference in any way convenient to you.

Tabs

Supplementary pages

You receive the most benefit from this manual if you first read through the table of contents for each chapter and then quickly scan the entire manual for an overview. This would be followed by a more detailed study of those chapters pertaining to your system. The examples to the right of the text are helpful in clarifying various concepts.

First, scan entire manual

This manual is organized in a top down manner -- more general and less complex material is covered first. Specific chapter contents are as follows:

Organization of manual

Chapter 1 is an introduction to a PDOS system.

Introduction

Chapter 2 describes the PDOS operation system in detail including the kernel, file manager, monitor, and floating point module.

PDOS system

Chapter 3 describes the monitor commands.

Monitor commands

Chapter 4 examines the assembly primitives of the PDOS kernel and file manager.

Assembly primitives

Chapter 5 discusses the PDOS editor and editor configurator.

Editor, editor configurator

Chapter 6 is divided into assembler and linker sections.

Assembler and linker

Chapter 7 provides detailed descriptions of the more common PDOS utilities.

Utilities

Chapter 8 gives a detailed description of the PDOS BIOS including the UARTs and read/write sector modules.

Secondary storage DSRs

The appendices give detailed descriptions of PDOS errors, I/O drivers, command summaries, and the window feature. They also include an index and a glossary.

Appendices

(1.1 HOW TO USE THIS MANUAL continued)

This manual is written in two columns. The left hand column functions much as the text of any book. The right hand column functions as an outline of the material in the left hand column and provides additional examples and explanations. Use it for quick reference to specific topics.

Page format

While much effort has gone into making this manual error free, some mistakes are still likely to be present. Your help in making the next edition better than the current one is appreciated. Please let Eyring know of any major mistakes or suggestions for chapters that need expansion.

Manual errors?

This manual assumes a moderate amount of computer hardware and software knowledge on your part. It also assumes familiarity with the MC68000 microprocessor. Such information is available in one or more of the following references:

Further reference

Motorola. 1984. MC68000 - 16/32-BIT MICROPROCESSOR PROGRAMMER'S REFERENCE MANUAL. Fourth Edition. Englewood Cliffs, N.J.: Prentice-Hall Inc.

Zarrella, John. 1981. MICROPROCESSOR OPERATING SYSTEMS. Suisun City, California: Microcomputer applications.

× **NOTATION**

The following notations are used throughout this manual:

$	Hexadecimal number. (e.g., $1FFF = decimal 8191.)
%	Binary number. (e.g., %1001101 = decimal 77.)
< >	Parameter used with a PDOS command or primitive. (e.g., DL <file name> indicates that the DL command requires a file name as a parameter.)
{ }	Optional. (e.g., SA <file name> {,<attributes>} indicates that the parameter <attributes> is optional.)
(Ax)	Indirect assembly addressing. (e.g., (A2) = Buffer refers to register A2 pointing to a buffer.)
[ ]	Control character or other key cap. (e.g., [CTRL-C] denotes a hexadecimal $03 character; [ESC] refers to the escape key.)

1.2 PDOS SYSTEM FEATURES

- Realtime, multi-user, multi-tasking
- Prioritized, round-robin scheduling
- Intertask communication and synchronization
- Full exception processing
- Fast interrupt task response
- Sequential, random, and shared file management
- Hardware independence
- 68000 layered design of kernel, file manager, monitor
- Configurable, modular, ROMable stand-alone support

Full Development System — Less than 24 KB

ROMable Modules
- Kernel
- File Manager
- Monitor
- Debugger

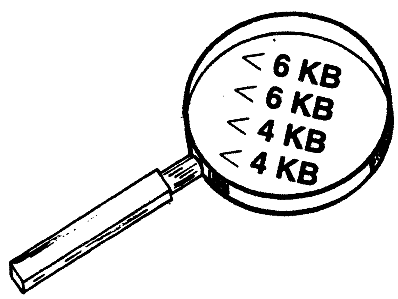

Figure 1.1 PDOS memory usage.

1.2.1 PDOS DESCRIPTION

Multi-user, multi-tasking

PDOS is a powerful multi-user, multi-tasking operating system developed by Eyring Research Institute, Inc., for the 32-bit Motorola 68000 processor family. You can use PDOS to design and develop scientific, educational, industrial, and business applications.

PDOS consists of a small, realtime, multi-tasking kernel layered by file management, and user monitor modules. The 6K byte kernel handles synchronization and control of events occurring in a realtime environment using semaphores, events, messages, mailboxes, and suspension primitives. All user console I/O as well as other useful conversion and housekeeping routines are included in the PDOS kernel.

```
|                                 |
|        USER APPLICATION         |
|---------------------------------|
|   MONITOR      |  FILE MANAGER  |
|----------------+----------------|
| BIOS = CPU, UARTS, R/W SECTOR   |
|.................................|
|          PDOS KERNEL            |
'---------------------------------'
```

File management module

The file management module supports named files with sequential, random, read only random, and shared access. Mass storage device independence is achieved through read and write logical sector primitives. The designer is relieved of realtime and task management problems as well as user console interaction and file manipulation so that efforts are concentrated on the application.

Secondary storage

PDOS is easily configured for any combination of large or small floppy disks, bubble memory devices, or Winchester mass storage devices. A wide variety of target system configurations are supported for fast development of memory-efficient, cost-effective end products.

1.2.2 PDOS FUNCTIONAL DESCRIPTION

PDOS kernel

PDOS KERNEL. PDOS is written in Motorola 68000 assembly language for fast, efficient execution. The small kernel handles multi-tasking, realtime clock, event processing, and memory management functions. Ready tasks are scheduled using a prioritized, round-robin method. The highest priority task, in the ready state, is always scheduled. Tasks with the same priority are scheduled in a round-robin fashion. A suspended task allows lower priority tasks to execute. The A-line ($A000) instruction interfaces over 100 system primitives to a user task.

Multi-tasking execution environment

MULTI-TASKING EXECUTION ENVIRONMENT. Tasks are the components comprising a realtime application. Each task is an independent program that shares the processor with other tasks in the system. Tasks provide a mechanism that allows a complicated application to be subdivided into several independent, understandable, and manageable modules. Realtime, concurrent tasks are allocated in 2K byte increments. There are no 64k byte boundary restrictions since the full 32-bit address space is available. Task system overhead is less than 2.5k bytes.

Intertask communication and synchronization

INTERTASK COMMUNICATION and SYNCHRONIZATION. Semaphores and events provide a low overhead facility for one task to signal another. Events indicate availability of a shared resource, timing pulses, or the occurrence of a hardware or software interrupt. Messages and mailboxes are used in conjunction with system lock, unlock, suspend, and event primitives. PDOS provides timing events that can be used in conjunction with desired events to prevent system lockouts. Other special system events signal character inputs and outputs.

Exception processing

EXCEPTION PROCESSING. PDOS handles all exception processing including interrupts, address errors, bus errors, illegal and unimplemented instructions, and privilege violations. Each task also has the option to process any or all 16 TRAP vectors, divide by zero, overflow check (TRAPV), and register out of bounds (CHK). System interrupts set the corresponding event and then can initiate a context switch. A high priority task waiting on that event would be immediately scheduled and begin executing.

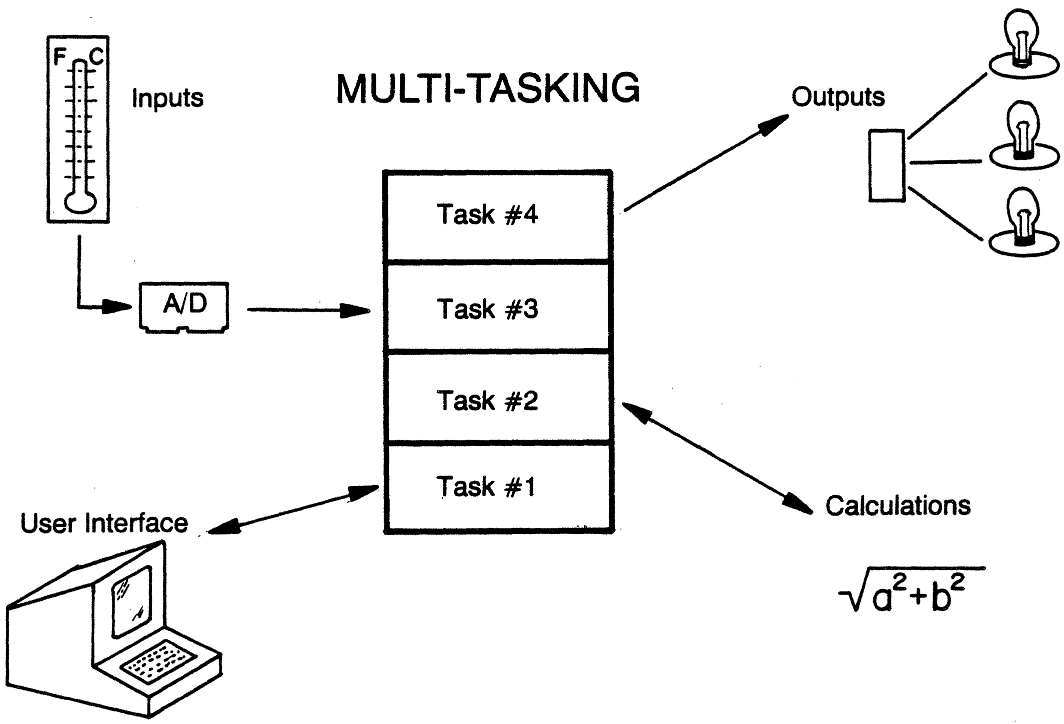

Figure 1.2 Multi-tasking PDOS.

(1.2.2 PDOS FUNCTIONAL DESCRIPTION continued)

Memory requirements

MEMORY REQUIREMENTS. PDOS is very memory efficient. The PDOS kernel, file manager, debugger, BIOS, and user monitor utilities require less than 16k bytes of memory plus an additional 6k bytes for system buffers and stacks. Most applications are both developed and implemented on the target system. Further memory reduction is achieved by linking the user application to a 6k byte PDOS kernel for a small, ROMable, stand-alone, multi-tasking module. For large system configurations, PDOS effectively addresses up to the 32-bit address space of the 68000 processor.

File management

FILE MANAGEMENT. The PDOS file management module provides sequential, random, read only, and shared access to named files on a secondary storage device. These low overhead file primitives use a linked, random access file structure and a logical sector bit map for allocation of secondary storage. No file compaction is ever required. Files are time stamped with date of creation and last update. Up to 127 files can be open simultaneously. Complete device independence is achieved through read and write logical sector primitives. Supported devices include floppies, bubble and battery backed-up memories, Winchester drives, and streaming tape drives.

Command Line Interpreter

COMMAND LINE INTERPRETER (CLI). The PDOS monitor calls the command line interpreter. The CLI parses the command line for multiple commands and parameters. Utilities such as append, define, delete, copy, rename, and show file are resident and execute without destroying current memory contents. Other functions in the PDOS monitor include setting the baud rate of a port; creating tasks; listing tasks, files and open file status; asking for help; setting file level, file attributes, interrupt mask, and system disk; and directing console output.

Interrupt management

INTERRUPT MANAGEMENT. The PDOS kernel handles user console, system clock, and other designated hardware interrupts. User consoles are interrupt-driven with character type-ahead. A task can be suspended pending a hardware or software event. Otherwise, a prioritized, round-robin scheduling of ready tasks occurs at 10 millisecond intervals.

(1.2.2 PDOS FUNCTIONAL DESCRIPTION continued)

PORTABILITY. PDOS gives software portability within 68000 systems through hardware independence of the system Basic Input/Output System (BIOS) module. All hardware functions such as read/write logical sector, clocks, mappers, and UARTs are conveniently isolated in this module for minimal customization to new 68000-based systems.

Portability

CUSTOMER SUPPORT. Numerous support utilities including screen editors, assembler, linker, macroprocessor, EPROMing, disk diagnostics and recovery, and disk cataloging are standard. Single stepping, multiple break points, memory snap shots debugger, task save and restore commands, and error trapping primitives in all high level languages are all provided to aid in program debugging. Upgrades are available with hotline service to system developers.

Customer support

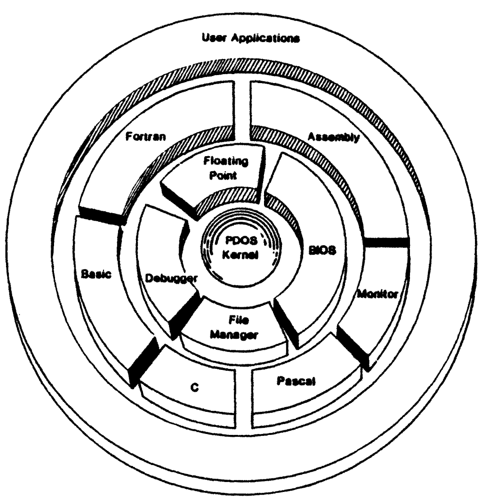

Figure 1.3 Extendable PDOS features.

1.3 PDOS DEMONSTRATION

This section gives a sample PDOS keyboard session. It is not intended as a start up procedure for new users, but rather, to give the flavor of the PDOS operating system environment. You will probably notice a number of differences from your system.

All entries are terminated by a carriage return [CR] unless otherwise specified. User entries are all underlined and indicated on those lines with a right bracket (>) in the left column.

Terminal session	Comments
SAGE II Startup Test [1.2] Booting from Floppy 68K PDOS Sage II Bootstrap	The SAGE boot EPROMs read a bootstrap program from sectors 0 through 3. These in turn load PDOS into memory from $800 to $9800. Execution begins at location $800.
Done Reading Header PDOS boot OK. Gooooooooo! PDOS/68000 R3.2 01-Nov-86 ERII, Copyright 1983-86 SAGE II BIOS 31-Jul-85 > DATE=00-???-00 11/10/86 > TIME=00:14:02 10 57	The PDOS banner lists the revision, date created, and the BIOS type. You then enter today's date and time. Terminate all entries with a [CR] unless otherwise specified. Date and time numbers can be separated by commas or spaces. Seconds are optional.
x>HE For further help, enter 'HE ' followed by one of the following:	The HE command reads the file called 'HLPTX' from the default disk. This lists the current help available to you.

```
      MONITOR {monitor command}      PDOS monitor commands
      FILE {file help}               List directory & file types
      BASIC {help string}            BASIC language help
      C {help string}                C language help
      FORTRAN {help string}          FORTRAN-77 help
      PASCAL {help string}           PASCAL language help
```

(1.3 PDOS DEMONSTRATION continued)

Further help concerning the PDOS monitor is listed by entering 'HE MONITOR'. This applies to all other help parameters.

```
x>HE MONITOR
 Current PDOS resident monitor commands:

 AC - Review procedure  GM - Get memory         RD - RAM disk
 AF - Append file       GO - Execute            RN - Rename file
 BP - Baud port         GT - Go to label        RS - Reset
 CF - Copy file         HE - Help               SA - Set file attributes
 CT - Create task       IA - If altered         SF - Show file
 DF - Define file       ID - Init date & time   SM - Send task message
 DL - Delete file       IF - Conditional        SP - Disk usage
 DM - Delete multiple   KM - Kill message       SU - Spool unit
 DN - Download file     KT - Kill task          SV - Save to file
 DT - Display time      LL - List levels        SY - System disk
 EE - Enable echo       LO - Load file          TF - Transfer files
 ER - List error        LS - List directory     TM - Transparent mode
 EV - Events            LT - List tasks         TP - Task priority
 EX - Basic             LV - Directory level    UN - Output unit
 FE - For every         MF - Make file          UP - Upload from port
 FM - Free memory       PB - Debugger           ZM - Zero memory
 FS - File slots        RC - Reset console
Hit <CR> to continue.....[CR]
```

Some help messages are paged and require a character from your terminal console to be entered to continue.

```
 Monitor command formats are as follows:

 AC <file>                        Review procedure file
 AF <file1>,<file2>               Append file
 BP {{-}<prt>,<rt>{,<ty>,<bs>}}   Baud port
 CF <file1>,<file2>               Copy file
 CT <cmd>,<sze>,<prity>,<prt>     Create task
 DF <file>{,<size>}               Define file
 DL <file>                        Delete file
 DM <filelist>                    Delete multiple
 DN <file>                        Download file
 DT                               Display time
 EE <0=no echo>                   Enable echo
 ER <error#>                      List error
 EV {{-}<event>}                  Events
 EX                               Basic
 FE <fl> or (<s>,<e>),<cmd>       For every
 FM {<kbytes>}                    Free memory
 FS                               File slots
 GM {<kbytes>}                    Get memory
 GO {<address>}                   Execute
 GT <label>                       Go to label
 HE {<list>}                      Help

 Hit <CR> to continue.....[CR]
```

(1.3 PDOS DEMONSTRATION continued)

```
 IA <file>.<command>            If altered
 ID                             Init date & time
 IF <str1>{=#<str2>}            Conditional
 KM <task#>                     Kill message
 KT {-}<task#>                  Kill task
 LL <filelist>                  List level
 LO <file>{,<start addr>}       Load file
 LS {<filelist>}{,<file>}       List directory
 LT                             List tasks
 LV {<level>}                   Directory level
 MF <file>                      Make file
 PB                             Debugger
 RC                             Reset console
 RD {{-}<unt>,<sze>,<adr>}      RAM disk
 RN <file1>,<file2>             Rename file
 RS {<disk>}                    Reset
 SA <file>{,<attribute>}        Set file attributes
 SF {-}<file>                   Show file
 SM {<task#>,<message>}         Send task message
 SP {<disk>}                    Disk usage
 SU <unit>{,<file> or <port#>}  Spool unit

Hit <CR> to continue.....[CR]

 SV <file>{,<sadr>,<eadr>}      Save to file
 SY {<disk>...}                 System disk
 TF <filelist>,<disk#>{,<flag>} Transfer files
 TM {{-}<port>}{,<break>}       Transparent mode
 TP {<task#>,}<priority>        Task priority
 UN {<unit>}                    Output unit
 UP {<port #>}{,<message>}      Upload from port
 ZM                             Zero memory
```

LT lists the currently executing tasks.

```
x>LT
Task   Prt Tm  Event   Map  Size   PC       SR   TB       EM        I U 1 2 4 8
 1/-1  100 1   127      0   32     0006854A 2000 00068000 00070000  0 1 0 0 0 0
 2/0   255 1   -128     0   2      00067D2E 0004 00067800 00068000  0 1 0 0 0 0
 3/0   64  1   100      0   32     0000339A 2004 0005F800 00067800  4 1 4 0 0 0
```

(1.3 PDOS DEMONSTRATION continued)

```
x>HE MONITOR LT
 Command: List tasks
  Format: LT

 List Task heading explanation:

        Task    {*=current}Task #/parent task #
        Prt     Task priority (1-255) (+ indicates SVF$ set)
        Tm      Task CPU tics (1 tic=10 ms)
        Event   Suspended event(s)
        Map     Task map constant
        Size    Task size (k bytes)
        PC      Program Counter
        SR      Status Register
        TB      Task control Block
        BM      Beginning of memory
        EM      End of memory
        I       Input port number
        1       Unit 1 port number
        2       Unit 2 port number
        4       Unit 4 port number
        8       Unit 8 port number
```

'HE MONITOR LT' explains the LT parameters. Help information is available on all monitor commands by typing 'HE MONITOR xx' where xx is any monitor command. Information about that monitor command will then be displayed.

```
x>BP
Port  Type  fwpi8dcs   Base     Rate  Task
#1    1     00000000  00F20001  9600   0
#2    2     00000001  00FF1000  9600
#3    2     00000000  00FF1040  9600
```

'BP' lists the currently installed ports.

```
x>EV
 00000000 00000000 00000000 0000FE00
Event=130  Delay=85
```

'EV' lists current event states.

```
x>DT
DATE=10-Nov-86
TIME=11:17:39
```

'DT' lists the current date and time.

```
x>RD
Disk=8
Size=255
Addr=00070000
```

'RD' lists the RAM disk parameters.

(1.3 PDOS DEMONSTRATION continued)

```
x>LKJLKJLHHJKJ
PDOS ERR 50 Illegal name
x>ASDFD
PDOS ERR 53 Not defined
```

PDOS attempts to explain error numbers.

```
x>LV
Level=1
20,2>SY
Disk=20,2
```

Other commands are resident in the monitor and readily available, such as list the directory level (LV) or default disk (SY).

There are 255 directory levels for each disk number. The current level is 1.

There are 255 different disk units. The current disk unit is 20 for working and file creation and 2 for reference.

```
x>SP
Files=8/48
Free=2490,2490
Used=1/8
```

The SP command outputs the number of files used out of those available, the number of free sectors on a disk, and the number of sectors used and the file directory size. The second parameter on 'Free' is the largest contiguous block of sectors on the disk. The 'Used' output is divided between the number of sectors actually used versus the number of sectors allocated to files from the disk bit map.

```
x>UN.FM.EV
Unit=3
x>FM.EV
Free=0
x>EV
 00000000 00000000 00000000 0000FE00
Event=130  Delay=9
```

Multiple commands are entered on the same line by separating the commands (along with any parameters for the command) with a period. As each command is executed, the command line is echoed again.

```
x>[CTRL-A]
UN.FM.EV
Unit=3
x>FM.EV
Free=0
x>EV
 00000000 00000000 00000000 0000FE00
Event=128  Delay=58
Event=130  Delay=97
```

The command line is saved and can be recalled by entering a [CTRL-A].

Events are used for task synchronization. Each event is a single bit. The system events (112-127) are generally set.

(1.3 PDOS DEMONSTRATION continued)

```
X 0>LS
  Disk=PDOS 3.2/0                       Files=13/128
  Lev  Name:ext     Type      Size      Sect   Date created     Last update
   1   CHAP00       TX C       4/4      0012  14:25 08-Aug-85 16:03 21-Aug-85
   1   CHAP01       TX +     141/141    0016  09:53 09-Aug-85 12:03 10-Feb-86
   1   CHAP02       TX C     216/216    00A2  12:11 20-Aug-84 10:57 27-Aug-85
   1   CHAP03       TX C     326/326    017A  12:11 20-Aug-84 14:20 26-Aug-85
   1   CHAP04       TX C     622/622    02C0  12:12 20-Aug-84 10:09 27-Aug-85
   1   CHAP05       TX C     120/120    052E  12:31 08-Aug-85 09:12 23-Aug-85
   1   CHAP06       TX C     363/363    05A6  11:00 08-Aug-85 12:42 26-Aug-85
   1   CHAP07       TX C     256/256    0711  15:10 23-Aug-85 18:10 23-Aug-85
   1   DEMO         AC C       2/2      0811  09:47 22-Jul-85 09:56 01-Aug-85
   1   PQ           EX C      33/33     0813  12:13 20-Aug-84 13:15 27-Sep-85
   1   TITLE        TX C**     4/4      0834  08:28 19-Jul-85 15:40 05-Aug-85
   1   TEMP         TX C       1/1      0838  09:51 23-Aug-85 14:10 23-Aug-85
   1   SEND         AC C*      1/1      0839  16:06 23-Aug-85 13:16 27-Sep-85
  Files=13          Used=2089/2089
```

The LS command lists a disk directory. Parameter defaults are the current disk and directory level. Each file is time stamped with date of creation and last update. The file size indicates the number of sectors actually used versus the number of sectors allocated to the file from the sector bit map. Hitting any key will pause the output listing. Pressing another key continues the listing. The [ESC] key terminates the output.

```
x>HE FILE
       FILE FS         Directory header explanation
       FILE FILELIST   PDOS file selection list definition
x>HE FILE FS
 List directory header explanations:

       LEV             File directory level
       NAME:EXT        File name:extension
       TYPE            File attribute (See below)
       SIZE            Sectors user/Sectors allocated
       SECT            Start sector number
       DATE CREATED    Time & date file defined
       LAST UPDATE     Time & date file was last altered
```

The 'HE FILE FS' command explains the heading definitions for the list directory command.

```
 Valid file types are as follows:

       AC = Procedure file             +  = Altered
       OB = 68000 object               C  = Contiguous
       SY = System file                *  = Delete protect
       TX = ASCII text                 ** = Write protect
       BN = Binary file
       EX = BASIC program
       BX = BASIC binary program
       DR = System I/O driver
```

The PDOS monitor uses the file type in controlling the file processing. A file typed as 'OB' contains 68000 tagged object and is loaded into task memory and executed. 'SY' or system files are handled similarly. 'EX' files are directed to the resident BASIC interpreter, loaded and executed.

(1.3 PDOS DEMONSTRATION continued)

```
x>HE FILE FILELIST
 A PDOS file selection list is defined as follows:

  <filelist> = {file}{:ext}{;level}{/disk}{/select...}

                {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
                {:ext} = 1 to 3 characters (:@=all,*=wild)
              {;level} = directory level (;@=all)
               {/disk} = disk number ranging from 0 to 255
             {/select} = /AC = Assign Console file
                         /BN = Binary file
                         /BX = PDOS BASIC token file
                         /EX = PDOS BASIC file
                         /OB = 68000 PDOS object file
                         /SY = System file
                         /TX = Text file
                         /DR = System I/O driver
                         /*  = Delete protected
                         /** = Delete and write protected
                         /Fmm-dy-yr = selects files with date of last change
                                      greater than or equal to 'mm-dy-yr'
                         /Tmm-dy-yr = selects files with date of last change
                                      less than or equal to <= 'mm-dy-yr'
```

'HE FILE FILELIST' explains how to select files using the PDOS file selection list used with many monitor commands.

```
x>FS
Slot Name                ST   SM      PT     SI    EOF    TN      BF      FLGS
 32  DOC;1/20           C104 0032 0000CD01 0023 0023/62 0000 0000CCAE 00000000
```

The 'FS' command lists open file slots.

```
x>HE MONITOR FS
 Command: File slots
  Format: FS

 List File Slots heading explanation:

        Slot    File slot #
        Name    File name ; directory level / disk
        ST      Channel status
        SM      Sector in memory
        PT      Channel buffer pointer
        SI      Current file sector index
        EOF     End-of-file sector index number / bytes in last sector
        TN      Task number which locked/opened the file
        BF      Channel buffer address+
        FLGS    Channel status flags (lock/shared/error)

        + A zero buffer address indicates the buffer has been
          rolled to disk.

Hit <CR> to continue.....[CR]
```

A file is accessed through a file slot or channel. This memory area contains all the status and pointers associated with an open file. The OPEN commands bind a file slot with a channel buffer where the file data is actually transferred.

(1.3 PDOS DEMONSTRATION continued)

Channel status is defined as follows:

x1xx	Sequential	xx80	Altered
x2xx	Random	xx04	Contiguous
x6xx	Shared random	xx02	Delete protected
xAxx	Read only random	xx01	Write protected

1xxx	Driver in channel
2xxx	Buffer locked in memory
4xxx	File altered
8xxx	Sector altered

From the channel status, you can tell what type of OPEN was done, whether the file has been altered, and its protection codes.

```
x>SF UPTIME
```

The SF command displays a file.

```
   100  REM UPTIME
   110  DIM D[1],M[2],T[1],W[2]
   120  DATE $D[0]: TIME $T[0]: T=TIC 0
   130  M=$D[0]: D=$D[0;4]: Y=$D[0;7]: C=19
   140  M1=M-2: Y1=Y: IF M1<1: M1=M1+12: Y1=Y-1: IF Y1<0: C=C-1
   150  W=INT[2.6*M1-0.19]+D+Y1+INT[Y1/4]-2*C+INT[C/4]
   160  W=INT[W-INT[W/7]*7+0.5]: IF W<0: W=W+7
   200  RESTORE W+1: READ $W[0]
   210  DATA "Sunday","Monday","Tuesday","Wednesday"
   220  DATA "Thursday","Friday","Saturday"
   230  RESTORE M: READ $M[0]
   240  DATA "January","February","March","April"
   250  DATA "May","June","July","August","September"
   260  DATA "October","November","December"
   300  PRINT "Today is ";$W[0];", ";$M[0];D;",";C*100+Y;
   310  PRINT ".  The time is ";$T[0];"."
   315  DC=86400*SYS[38]: HC=3600*SYS[38]: MC=60*SYS[38]
   320  DAY=INT[T/DC]: T=T-DAY*DC
   330  HRS=INT[T/HC]: T=T-HRS*HC
   340  MIN=INT[T/MC]: T=T-MIN*MC
   350  SEC=INT[T/SYS[38]]
   360  PRINT "PDOS has been up for";
Strike any key...[CR]
   370  IF DAY: PRINT DAY;" days,";
   380  IF HRS: PRINT HRS;" hours,";
   390  IF MIN: PRINT MIN;" minutes, and";
   400  PRINT SEC;" seconds.";
     . . .
```

(1.3 PDOS DEMONSTRATION continued)

```
x>UPTIME
  Today is Monday, February 10, 1986.  The time is 12:09:04.
  PDOS has been up for 2 hours, 8 minutes, and 10 seconds.

                   February 1986
                Su Mo Tu We Th Fr Sa
                                   1
                 2  3  4  5  6  7  8
                 9 10 11 12 13 14 15
                16 17 18 19 20 21 22
                23 24 25 26 27 28
```

The UPTIME program may be executed simply by entering the file name.

```
0>LT
  Task  Prt Tm  Event   Map Size    PC      SR     TB       EM      I U 1 2 4 8
  *0/0  64  1            0   752  00002004 0000 0000C000 000C8000  1 1 1 2 0 0
   2/0  50  1   99       0   50   00001D1C 2004 000C8000 000D4800  3 1 3 0 0 0

x>CT ,100,,2
  *Task #1
```

A new task (or user) is created with the CT command. The task number is assigned by PDOS. Here, a new task of 100 Kbytes of memory on port 2 is created.

```
x>LT
  Task  Prt Tm  Event   Map Size    PC      SR     TB       EM      I U 1 2 4 8
  *0/0  64  1            0   752  00002004 0000 0000C000 000C8000  1 1 1 2 0 0
   1/0  64  1            0   100  000019EC 2004 000D4800 000ED800  2 1 2 0 0 0
   2/0  50  1   99       0   50   00001D1C 2004 000C8000 000D4800  3 1 3 0 0 0

x>CT ,50,50,3
  *Task #3
```

Additional tasks can be created. This one is 50 k bytes in size and has a priority of 50. Its I/O is through port 3.

```
x>LT
  Task  Prt Tm  Event   Map Size    PC      SR     TB       EM      I U 1 2 4 8
  *0/0  64  1            0   702  00002004 0000 0000C000 000BB800  1 1 1 2 0 0
   1/0  64  1            0   100  000019EC 2004 000D4800 000ED800  2 1 2 0 0 0
   2/0  50  1   99       0   50   00001D1C 2004 000C8000 000D4800  3 1 3 0 0 0
   3/0  50  1            0   50   000019EC 2004 000BB800 000C8000  0 1 3 0 0 0

x>KT 3
```

Tasks are just as easily removed from the task list with the KT command.

```
x>LT
  Task  Prt Tm  Event   Map Size    PC      SR     TB       EM      I U 1 2 4 8
  *0/0  64  1            0   702  00002004 0000 0000C000 000BB800  1 1 1 2 0 0
   1/1  255 1            0   100  000019EC 2004 000D4800 000ED800  2 1 2 0 0 0
   2/0  50  1   99       0   50   00001D1C 2004 000C8000 000D4800  3 1 3 0 0 0

x>FM
  Free=50
```

After a task is killed, its memory is allocated in the memory bit map. The FM command lists any memory available to the current task.

(1.3 PDOS DEMONSTRATION continued)

The GM command allows this memory to be recovered. Any or all memory is easily allocated to your task.

```
x>GM 40
x>FM
  Free=10
x>LT
  Task   Prt Tm  Event   Map Size    PC       SR    TB       EM       I U 1 2 4 8
  *0/0   64  1            0   702   00002004 0000 0000C000 000BE000  1 1 1 2 0 0
   1/1   255 1            0   100   000019EC 2004 000D4800 000ED800  2 1 2 0 0 0
   2/0   50  1   99       0   50    00001D1C 2004 000C8000 000D4800  3 1 3 0 0 0
x>GM
x>FM
  Free=0
x>LT
  Task   Prt Tm  Event   Map Size    PC       SR    TB       EM       I U 1 2 4 8
  *0/0   64  1            0   702   00002004 0000 0000C000 000C8000  1 1 1 2 0 0
   1/1   255 1            0   100   000019EC 2004 000D4800 000ED800  2 1 2 0 0 0
   2/0   50  1   99       0   50    00001D1C 2004 000C8000 000D4800  3 1 3 0 0 0
```

The MF or make file command allows you to create a file directly from your keyboard console. Assembly language development is very easy because all operating system calls are supported by the assembler. This program simply prints a message and returns to the PDOS monitor.

```
x>MF #PRGM:SR
*       PRGM:SR   09/20/83
*
START   XPMC MES01        ;OUTPUT MESSAGE
        XEXT              ;DONE
*
MES01   DC.B $0A,$0D,'IT WORKS!!!!',0
        END  START
[ESC]
```

Let's look at it again to check its syntax.

```
x>SF PRGM:SR
START   XPMC MES01        ;OUTPUT MESSAGE
        XEXT              ;DONE
MES01   DC.B $0A,$0D,'IT WORKS!!!!',0
        END  START
```

The assembler is called and the object is directed to a new file called PRGM. The assembly listing goes to the file LIST.

```
0>MASM PRGM:SR,#PRGM,#LIST
68K PDOS Assembler R3.2  01-Nov-86
ERII, Copyright 1983-86
SRC=PRGM:SR
OBJ=#PRGM
LST=#LIST
ERR=
XRF=
END OF PASS 1
END OF PASS 2
```

Enter the program name to execute the file.

```
x>PRGM
IT WORKS!!!!
```

(1.3 PDOS DEMONSTRATION continued)

```
x>SF LIST

                                            68K PDOS Assembler 01-Nov-86
 PAGE: 1         09:56 08/01/85             FILE: PRGM:SR,PDOS 3.2 SYSTEM


  1                                *        PRGM:SR   09/20/83
  2                                *
  3  0/00000000:A08C0004           START    XPMC MES01        ;OUTPUT MESSAGE
  4  0/00000004:A00E                        XEXT              ;DONE
  5                                *
  6  0/00000006:0A0D495420574F52 MES01      DC.B $0A,$0D,'IT WORKS!!!!',0
  7             4B532121212100
  8  0/00000015:       0/00000000           END  START

                                            68K PDOS Assembler 01-Nov-86
 PAGE: 2         09:56 08/01/85             FILE: PRGM:SR,PDOS 3.2 SYSTEM


 DEFINED SYMBOLS:

 MES01         0/00000006   START         0/00000000

 EXTERNAL DEFINITIONS: NONE
 Strike any key...

 EXTERNAL REFERENCES: NONE

 UNDEFINED SYMBOLS: NONE

 UNREFERENCED SYMBOLS: NONE
```

(1.3 PDOS DEMONSTRATION continued)

```
x>LS /8
  Disk=SY$DSK/8                         Files=0/32
  Lev  Name:ext     Type      Size      Sect  Date created    Last update

x>TF P@:@,8,A
  Transfer PRGM:SR;1/0
  Transfer PRGM;1/0
  Transfer PRINTQ:BGR;1/0
  Transfer PRINTRX;1/0
  Transfer PRINTS:BGR;1/0
```

The TF command transfers multiple files from one disk to another.

```
x>LS /8
  Disk=SY$DSK/8                         Files=5/32
  Lev  Name:ext     Type      Size      Sect    Date created    Last update
   1   PRGM:SR         C       1/1      0005  14:25 08-Aug-85 16:03 21-Aug-85
   1   PRGM         OB C       1/1      0006  09:53 09-Aug-85 12:03 10-Feb-86
   1   PRINTQ:BGR   EX C     34/34      0007  12:13 20-Aug-84 10:57 27-Aug-85
   1   PRINTRX      EX C     35/35      0029  12:13 20-Aug-84 14:20 26-Aug-85
   1   PRINTS:BGR   EX C     34/34      004C  12:13 19-Jul-85 15:40 05-Aug-85

x>DM @:@;@/8
  Delete PRGM:SR;1/8? (Y/N/A)A
  Delete PRGM;1/8
  Delete PRINTQ:BGR;1/8
  Delete PRINTRX;1/8
  Delete PRINTS:BGR;1/8
x>_
```

The DM command deletes multiple files from a disk directory.

OPERATING SYSTEM

CHAPTER 2

PDOS SYSTEM OVERVIEW

The PDOS operating system is described here in detail. There are four main sections of PDOS; namely, the kernel, BIOS, file management module, and monitor.

2.1 PDOS KERNEL

The PDOS kernel is the multi-tasking, realtime nucleus of the PDOS operating system. Tasks are the components comprising a realtime application. It is the main responsibility of the kernel to see that each task is provided with the support it requires in order to perform its designated function.

The main responsibilities of the PDOS kernel are the allocation of memory and the scheduling of tasks. Each task must share the system processor with other tasks. The operating system saves the task's context when it is not executing and restores it again when it is scheduled. Other responsibilities of the PDOS kernel are maintenance of a 24-hour system clock, task suspension and rescheduling, event processing, character buffering, and other support functions.

PDOS kernel:

1. Multi-tasking, multi-user scheduling
2. System clock
3. Memory allocation
4. Task synchronization
5. Task suspension
6. Event processing
7. Character I/O including buffering
8. Support primitives

2.1.1 PDOS TASK

A PDOS task is defined as a program entity which can execute independently of any other program if desired. It is the most basic unit of software within an operating system. A user task consists of an entry in the PDOS task queue, task list, and a task control block with user program space.

The task queue and list are used by the PDOS kernel to schedule tasks. A task queue entry consists of a task priority and a task number. The list is ordered with the highest priority entry first. A task list entry consists of a priority, task time, spawned task number, task control block pointer, task map constant, and two suspended event registers. The task number is assigned according to its entry position.

The first $500 (hex) bytes of a task are the task control block. This block of memory consists of buffers and parameters peculiar to the task. The 68000 address register A6 points to the status block when the user program is first entered. The task parameters may be referenced by a user program but care must be taken that PDOS is not crashed! The task control block variables are displacements beyond register A6 and are defined in FIGURE 2.1.

```
                                        Memory
Task Queue  Task List
----------  ---------                   Task #0
  100/#1     Task #0--->--->--->.---------.
   64/#0     Task #1----v       |  Task   |
   64/#2     Task #2    v       | Control |
   .....      ....      v       |  Block  |
                        v       |---------|
                        v       |  User   |
                        v       | Program |
                        v       |  Space  |
                        v          .....
                        v       |---------|
                        v       |         |
                        v
                        v
                        v               Task #1
                        '--->--->.---------.
                                 |         |
                                    .....
```

(2.1.1 PDOS TASK continued)

The user program space begins immediately following the task control block. Position independent 68000 object programs or BASIC tokens are loaded into this area for execution. Task memory is allocated in 2k byte increments. The total task overhead is $500 or 1280 bytes. This leaves $300 or 768 bytes available for a user program and user stack in a minimal 2k byte task.

Task overhead = $500 (hex) bytes + user stack

From the time a task is coded by a programmer until the task is destroyed, it is in one of four task states. Tasks move among these states as they are created, begin execution, are interrupted, wait for events, and finally complete their functions. These states are defined as follows:

4 task states:

1. Undefined
2. Ready
3. Running
4. Suspended

1. Undefined	A task is in this state before it is loaded into the task list. It can be a block of code in a disk file or stored in memory.
2. Ready	When a task is loaded in memory and entered in the task queue and list but not executing or suspended, it is said to be ready.
3. Running	A task is running when scheduled by the PDOS kernel from the task list.
4. Suspended	When a task is stopped pending an event external to the task, it is said to be suspended. A suspended task moves to the ready or running state when the event occurs.

A task remains undefined until it is made known to the operating system by making an entry in the task queue. Once entered, a task immediately moves to the ready state which indicates that it is ready for execution. When the task is selected for execution by the scheduler, it moves to the run state. It remains in the run state until the scheduler selects another task or the task requires external information and suspends itself until the information is available. The suspended state greatly enhances overall system performance.

2.1.2 MULTI-TASKING

PDOS defaults to allow 32 independent tasks to reside in memory and share CPU cycles. Each task contains its own task control block and thus executes independently of any other task. A task control block consists of buffers, pointers, and a PDOS scratch area. By changing the 'NT' parameter in MSYRAM and other parameters, PDOS can be configured to handle up to 128 tasks.

Defaults to 32 independent time-shared tasks

128 tasks can be handled.

Four parameters are required for any new task generation. These are:

1) A task priority. The range is from 255 (highest priority) to 1 (lowest priority).

255 = Highest priority
1 = Lowest priority

2) Tasking memory. Memory is allocated to a task in 2k byte increments. The first $500 bytes is assigned the task TCB.

Task memory

3) An I/O port. Input ports are unique while many tasks may share the same output port for task console communication.

I/O port

4) A task command. This may be in the form of several monitor commands or a memory address to begin executing.

Command

Each of the above requirements defaults to a system parameter. Task priority defaults to the parent task's priority. Default memory allocation is 32k bytes and default console port is the phantom port.

Task defaults

If a task command is not specified, the new task reverts to the PDOS monitor. However, if no input is possible (i.e. port 0 or input already assigned), then the new task immediately kills itself. This is very useful since tasks automatically kill themselves as they complete their assignments (remove themselves from the task list and return memory to the available memory pool).

Automatic task termination

A task entry in the task list consists of a task number designation, parent task number, time interval, task priority, memory map constant, task control block pointer, and two event registers. Swapping from one task to the next is done when the task interval timer decrements to zero, during an I/O call to PDOS, or when an external event causes a context switch. The task interval timer decrements by one every ten milliseconds (or as defined in the system BIOS module).

Task entry in task list

(2.1.2 MULTI-TASKING continued)

Task memory allocation

Any task may spawn another task. Memory for the new task is allocated in 2k byte blocks from a pool of available memory. If no memory is free, the spawning task's own memory is used and the parent task's memory is reduced in size by the amount of memory allocated to the new task. It is important to note that some assembly coded programs and all high level language programs use both the low and high addresses of the task memory. To prevent memory loss from a task and program failure, it is necessary to allocate enough memory to the free memory pool before creating a new task under program control. Otherwise, the task may give up its variable space or stack to the spawned task.

Memory bit map

PDOS maintains a memory bit map to indicate which segments of memory are currently in use. Allocation and deallocation are in 2k byte increments. When a task is terminated, the task's memory is automatically deallocated in the memory bit map and made available for use by other tasks.

Multi-user system

"Multi-user" refers to spawning new tasks for additional operators. Each new task executes programs or even spawns additional tasks. Such tasks are generated or terminated as needed. Task 0 is referred to as the system task and cannot be terminated.

Figure 2.1 shows the task control block.

WARNING: Although the locations of the task control block are made available to the user, you must be cautious when using these locations. Many PDOS primitives use these locations to perform their functions and any location may change at any time as a result of these PDOS calls. The same TCB format has mostly been retained throughout PDOS revisions; however, that may not always be the case and the TCB may be modified significantly.

```
TASK > |-------|                 Task Status Control Definitions
       |       |
       |   T   |         0(A6) = 256 byte user buffer
       |       |       $100(A6) = CLB$    - 82 byte monitor command line buffer
       |   C   |       $150(A6) = MWB$    - 32 byte monitor work buffer
       |       |       $170(A6) = MPB$    - monitor parameter buffer
       |   B   |       $3B0(A6) = TSP$.L - task stack pointer
       |       |       $3B4(A6) = KIL$.L - kill self address
       |       |       $3B8(A6) =         - Reserved
       |-------|       $3BC(A6) = SVF$.B - save 68881 registers flag
       |       |       $3BE(A6) = TRP$    - user TRAP vectors
       |       |       $3FE(A6) = ZDV$.L - zero divide trap
         .....         $402(A6) = CHK$.L - CHCK instruction trap
                       $406(A6) = TRV$.L - TRAPV instruction trap
                       $40A(A6) = TRC$.L - trace vector
                       $40E(A6) = FPA$.8 - floating point accumulator
                       $416(A6) = FPE$.L - fp error processor address
                       $41A(A6) = CLP$.L - command line pointer
                       $41E(A6) = BUM$.L - beginning user memory
                       $422(A6) = EUM$.L - end user memory
               |\      $426(A6) = EAD$.L - entry address
      ---------' \     $42A(A6) = IMP$.L - assigned input message pointer
      Task Control\    $42E(A6) = ACI$.W - assigned input file ID
          Block   /    $432(A6) = LEN$.W - last error number
      ---------. /     $434(A6) = SFI$.W - spooling unit file ID
               |/      $436(A6) = FLG$.W - task flags
                       $437(A6) = SLV$.B - directory level
                       $438(A6) = FEC$.B - file expansion count
                       $43A(A6) = CSC$.W - clear screen character(s)
                       $43C(A6) = PSC$.W - position cursor characters
                       $43E(A6) = SDS$.B - alternate system disk(s)
                       $441(A6) = SDK$.B - system disk
                       $442(A6) = EXT$.L - XEXT address
                       $446(A6) = ERR$.L - XERR address
                       $44A(A6) = CMD$.B - command line delimiter
                       $44B(A6) = TID$.B - task ID
                       $44C(A6) = ECF$.B - echo flag
                       $44D(A6) = CNT$.B - output column counter
                       $44E(A6) = MMF$.B - memory modified flag
                       $44F(A6) = PRT$.B - input port #
                       $450(A6) = SPU$.B - spooling unit mask
                       $451(A6) = UNT$.B - output unit mask
                       $452(A6) = U1P$.B - unit 1 port #
                       $453(A6) = U2P$.B - unit 2 port #
                       $454(A6) = U4P$.B - unit 4 port #
         .....         $455(A6) = U8P$.B - unit 8 port #
       |       |       $456(A6) =         - reserved
       |       |       $458(A6) = TWO$.W - monitor word temps
       |       |       $45A(A6) = TW1$.W - TWO-TW2 used by level
       |       |       $45C(A6) = TW2$.W -    2 primitives
       |       |       $45E(A6) =         - reserved
       |_______|       $470(A6) =         - debugger parameters
       |       | <<<<< $500(A6) <<<<<<< USER PROGRAM
```

FIGURE 2.1 TASK CONTROL BLOCK

2.1.3 SYSTEM SERVICES

System services

System services are those functions that a task requires of the operating system while entered in the task list. These requirements range from timing and interrupt handling to task coordination and resource allocation.

Time keeping facilities

PDOS provides many time-oriented functions which key off of the system hardware interval timer. The current time of day and date are maintained with fine adjustment parameters. A 32-bit counter is used for various delta time functions such as task scheduling and event delays.

Interrupts

Hardware interrupts are processed by the kernel BIOS or passed to user tasks. Tasks can be suspended pending the occurrence of an interrupt and then be rescheduled when the interrupt occurs. Interrupts such as the interval timer and character input or output are handled by the kernel itself.

Task coordination

Task coordination is an integral part of realtime applications since many functions are too large or complex for any single task. The PDOS kernel uses common or shared data areas, called mailboxes, along with a table of preassigned bit variables, called events, to synchronize tasks. A task can place a message in a mailbox and suspend itself on an event waiting for a reply. The destination task is signaled by the event, looks in the mailbox, responds through the mailbox, and resets the event signaling the reply.

System resources

System resources include the processor itself, system memory, and support peripherals. The PDOS kernel provides primitives to create and delete tasks from the task list. Memory is allocated and deallocated as required. Peripherals are generally a function of the file manager but are assigned and released via system events. Device drivers coordinate related I/O functions, interrupts, and error conditions. All of these functions are available to user tasks and thus tasks may spawn tasks and dynamically control their operating environment.

Support utilities

Other support utilities contained within the PDOS kernel include number conversion, command line decoding, date and time conversions, and message processing routines. Facilities are also provided for locking a task in the run state during critical code execution.

2.1.4 PDOS CHARACTER I/O

The flow of character data through PDOS is the most visible function of the operating system. Character buffering or type-ahead assures the user that each keyboard entry is logged, even when the application is not looking for characters. Character output is normally through program control (polled I/O).

Interrupt driver character type-ahead

Program control output

Inputs and outputs are through logical port numbers. A logical port is bound to a physical UART (Universal Asynchronous Receiver / Transmitter) by the baud port commands. Only one task is assigned to an input port at any one time while many tasks may share the same output port. It is then the responsibility of each task to coordinate all outputs.

Inputs and outputs through logical ports

PDOS CHARACTER INPUT

PDOS character inputs come from four sources: 1) user memory; 2) a PDOS file; 3) a polled I/O driver; or 4) a system input port buffer. The source is dictated by input variables within the task control block. Input variables are the Input Message Pointer (IMP$(A6)), Assigned Console Input (ACI$(A6)), and input port number (PRT$(A6)).

Character inputs:

1. User memory
2. PDOS disk file
3. PDOS I/O device driver
4. System input port buffer

When a request is made by a task for a character and IMP$(A6) is nonzero, then a character is retrieved from the memory location pointed to by IMP$(A6). IMP$(A6) is incremented after each character. This continues until a null byte is encountered, at which time IMP$(A6) is set to zero.

```
        OPT     PDOS          ;GET TCB VARIABLES
        LEA.L   CMMD(PC),A1   ;POINT TO COMMAND
        MOVE.L  A1,IMP$(A6)   ;SET INPUT POINTER
        ....
CMMD    DC.B    'MESSAGE',0
        EVEN
```

If IMP$(A6) is zero and ACI$(A6) is nonzero, then a request is made to the file manager to read one character from the file assigned to ACI$(A6). The character then comes from a disk file or an I/O device driver. This continues until an error occurs (such as an END-OF-FILE) at which time the file is closed and ACI$(A6) is cleared.

```
        OPT     PDOS          ;GET TCB VARIABLES
        LEA.L   FILEN(PC),A1  ;POINT TO FILE NAME
        XSOP                  ;OPEN FILE
          BNE.S ERROR
        MOVE.W  D1,ACI$(A6)   ;SET CONSOLE INPUTS
        ....
FILEN   DC.B    'INDATA',0
        EVEN
```

If both IMP$(A6) and ACI$(A6) are zero, then the logical input port buffer selected by PRT$(A6), is checked for a character. If the buffer is empty, then the task is automatically suspended until a character interrupt occurs.

PDOS character input flow is summarized by Figure 2.2.

```
        OPT     PDOS          ;GET TCB VARIABLES
        MOVEQ.L #3,D1         ;READ CHARACTERS FROM
        MOVE.B  D1,PRT$(A6)   ;  PORT #3
        ....
```

```
                                                  |    |
      XPMC                     .<<<<<<<<<<<<<<<   | 7  | UNT$(A6)
      XPLC                     v                  |    |
      XPBC                     v                  |    |
       v                       v  .<<<<<<<<<<<<   | 4  | SPU$(A6)
       v                       v  v               |    |
       v  1. SPOOLing UNIT     v  v               |    |
   D   v                  _____v__v__             |    | SFI$(A6)
   A   v                 /           \            |    |/
   T   v>>>>>>>>>>>>>>><IF (UNT$^SPU$)>>>>>>>>>>|SFI$|>>>>>>>> [PDOS FILE]
   A   v                 \_____v__v__/            |    |            or
       v                       v  v               |    |        [I/O DRIVER]
   F   v                 ._____v__v___.           |    |
   L   v                 |unt=-SPU$^UNT$|         |    |
   O   v                 '-------v------'         |    |
   W   v                       v                  |    |
       v  2. Output UNIT 1     v                  |    |
   v   v                  _____v_____             |    | U1P$(A6)
   v   v                 /           \            |    |/
       v>>>>>>>>>>>>>>>< IF (unt$^1)  >>>>>>>>>>| 1  |>>>>>>>> [Port #1 UART]
       v                 \_____v_____/            |    |
       v                       v                  |    |
       v  3. Output UNIT 2     v                  |    |
       v                  _____v_____             |    | U2P$(A6)
       v                 /           \            |    |/
       >>>>>>>>>>>>>>>>< IF (unt$^2)  >>>>>>>>>>| 3  |>>>>>>>> [Port #3 UART]
       v                 \_____v_____/            |    |
       v                       v                  |    |
       v  4. Output UNIT 4     v                  |    |
       v                  _____v_____             |    | U4P$(A6)
       v                 /           \            |    |/
       >>>>>>>>>>>>>>>>< IF (unt$^4)  >>>>>>>>>>| 2  |>>>>>>>> [Port #2 UART]
       v                 \_____v_____/            |    |
       v                       v                  |    |
       v  5. Output UNIT 8     v                  |    |
       v                  _____v_____             |    | U8P$(A6)
       v                 /           \            |    |/
       >>>>>>>>>>>>>>>>< IF (unt$^8)  >>>>>>>>>>| 0  |>>>>>>>> [Phantom port]
                         \___________/            |    |
                                                  |    |
```

```
Notes:   UNIT 1  = (-SPU$ ^ UNT$) ^ 1
         UNIT 2  = (-SPU$ ^ UNT$) ^ 2
         UNIT 4  = (-SPU$ ^ UNT$) ^ 4
         UNIT 8  = (-SPU$ ^ UNT$) ^ 8
       PDOS FILE = (SPU$ ^ UNT$)
```

FIGURE 2.3 PDOS CHARACTER OUTPUTS

2.1.5 EVENTS

Tasks communicate by exchanging data through mailboxes. Tasks synchronize with each other through events. Events are single bit flags that are global to all tasks.

Events synchronize tasks

There are four types of event flags in PDOS: software, software resetting, system, and local. System events are further divided into output, input, timing, driver, and system resource events. System events are predefined software resetting events that are set during PDOS initialization. Event 128 is local to each task and is used as a delay event.

4 types of event flags:

1-63 = Software
64-80 = Software resetting
81-127 = System
128 = Local to task

1) 1-63 Events 1 through 63 are software events. They are set and reset by tasks and not changed by PDOS task scheduling. A task can suspend itself pending a software event and then be rescheduled when the event is set. One task must take the responsibility of resetting the event for the sequence to occur again.

1-63 = Software events

2) 64-80 Events 64 through 80 are like the normal software events except that PDOS resets the event whenever a task suspended on that event is rescheduled. Thus, one and only one task is rescheduled when the event occurs.

These events are set and reset by the Send Message Pointer (XSMP) and Get Message Pointer (XGMP) primitives.

64-80 = Software resetting events

3) 81-95 Events 81 through 95 correspond to output ports 1 through 15. A task suspends itself on an output event after transmitting a character through a UART. When the transmit character complete interrupt occurs, the event is set and the corresponding suspended task continues execution.

NOTE: Output port events are only supported though the xxBIOSU routines. See your Installation and Systems Management guide for implementation details.

81-95 = Output port events

```
1. MEMORY MESSAGE                 TASK CONTROL BLOCK
                                       |     |
   MSG   DC.B 'HELLO',0 >>>>>>>>>>>>>> |(MSP)| IMP$(A6)
                                       |     |   \
                                       |     |    \
                                       |     |     \
2. PDOS FILE W/TYPE=AC                 |     |      \
                                       |     |       \
       DO:AC >>> [CHANNEL BUFFER]      |     |        \
                          v            |     |         \
                          v            |     |          \
                          +>>>>>>>>>>> |FILID| ACI$(A6)>>+>> INPUT
                          ^            |     |          /
3. PDOS I/O DRIVER        ^            |     |         /
                          ^            |     |        /
       TTI >>>>> [POLLED I/O DRIVER]   |     |       /
                                       |     |      /
4. SYSTEM INPUT PORT BUFFER            |     |     /
                                       |     |    /
  KEYBOARD                 INPUT PORT  |     |   /
     v       UART.base      BUFFERS    |     |  /
     v       ._______.     ._______.   |     | /
     v       |adr 1  |     |BUF #1|    |     |/
    UART >>> |adr 2  | >>  |BUF #2| >> |  2  | PRT$(A6)
             |adr 3  |     |BUF #3|    |     |
             |adr 4  |     |BUF #4|    |     |
              .....         .....        ....
             |adr 15|      |BUF #F|    |     |
             '------'      '------'    |     |
```

NOTES: 1) UART.base binds a physical UART to a logical port number.

2) UART baud rate, address, and type are defined by the 'BP' and 'BAUD' commands (XBCP primitive).

3) XGCC gets characters from input port buffers only.

FIGURE 2.2 PDOS CHARACTER INPUTS

(2.1.4 PDOS CHARACTER I/O continued)

PDOS CHARACTER OUTPUTS

PDOS character outputs are directed to various destinations according to output variables in the task control block. Output variables are the output unit (UNT$(A6)), spooling unit (SPU$(A6)), spooling file ID (SFI$(A6)), and output port variables U1P$(A6), U2P$(A6), U4P$(A6), and U8P$(A6). The output unit selects the different destinations. (This is NOT to be confused with disk unit numbers.)

When an output primitive is called, the task output unit is ANDed with the task spooling output unit. If the result is nonzero, then the character is directed to the file manager and written to the file specified by SFI$(A6). The output unit is then masked with the complement of the spooling unit and passed to the UART character output processor.

Units 1, 2, 4, and 8 are special output numbers. Unit 1 is the console output port assigned when the task was created. Units 2, 4, and 8 are an optional output ports that correspond to TCB variables U2P$, U4P$, and U8P$. They are assigned by the spool unit command (.SU) or baud port command.

If the 1 bit (LSB) is set in the masked output unit (UNT$(A6)), then the character is directed to port U1P$(A6). Likewise, if bits 2, 3, or 4 are is set in the masked output unit, then the character is output to the U2P$(A6), U4P$(A6), or U8P$(A6) ports.

In summary, the bit positions of the output unit are used to direct output to various destinations. More than one destination can be specified. Bits 1 through 4 are predefined according to U1P$, U2P$, U4P$ and U2P$ variables within the task control block. Other unit bits are used for outputs to files and device drivers. Thus, if SPU$(A6)=4 and UNT$(A6)=7, then output would be directed to the file manager via SFI$(A6) and to two UARTs as specified in U1P$(A6) and U2P$(A6). (See Figure 2.3.)

```
        SPU$(A6) = 0000 0000 0000 0100
        UNT$(A6) = 0000 0000 0000 0111
                                  ///
                                 ///
              File SFI$(A6)____///
              Port U2P$(A6)____//
              Port U1P$(A6)____/
```

```
        OPT      PDOS           ;GET TCB VARIABLES
        LEA.L    FILEN(PC),A1   ;GET FILE NAME
        XSOP                    ;OPEN FILE
          BNE.S  ERROR
        MOVE.W   D1,SFI$(A6)    ;SET SPOOL FILE ID
        MOVEQ.L  #0,D1          ;CLEAR COUNTER
        MOVE.B   #4,SPU$(A6)    ;SET SPOOL UNIT TO 4
*
LOOP    MOVE.B   D1,UNT$(A6)    ;SELECT UNIT
        XCBM     MES01          ;CONVERT NUMBER
        XPLC                    ;OUTPUT MESSAGE
        ADDQ.W   #1,D1          ;INCREMENT D1
        CMPI.W   #8,D1          ;8 TIMES?
          BLT.S  LOOP           ;N
        ....                    ;Y

FILEN   DC.B     'OFILE',0      ;OUTPUT FILE NAME
MES01   DC.B     'OUTPUT MESSAGE #'.0
        EVEN

UNIT 1 =         OUTPUT MESSAGE #1
                 OUTPUT MESSAGE #3
                 OUTPUT MESSAGE #5
                 OUTPUT MESSAGE #7

UNIT 2 =         OUTPUT MESSAGE #2
                 OUTPUT MESSAGE #3
                 OUTPUT MESSAGE #6
                 OUTPUT MESSAGE #7

OFILE =          OUTPUT MESSAGE #4
                 OUTPUT MESSAGE #5
                 OUTPUT MESSAGE #6
                 OUTPUT MESSAGE #7
```

(2.1.5 EVENTS continued)

4) 96-111 Events 96 through 111 correspond to input ports 0 through 15. A task suspends itself on an input event if a request is made for a character and the buffer is empty. Whenever a character comes into an interrupt driven input port buffer, the corresponding event is set.

96-111 = Input port events

5) 112-115 Events 112 through 115 are timing events and are set automatically by the PDOS clock module according to intervals defined in the PDOS Basic I/O module (BIOS). Event 112 is measured in tics, while events 113, 114, and 115 are in seconds. The maximum time interval for event 112 is 497 days. Events 113, 114, and 115 have a maximum interval of 4,294,967,300 seconds or approximately 136 years. A task suspended on one of these events is regularly scheduled on a tic or second boundary.

112 = 1/5 second event
113 = 1 second event
114 = 10 second event
115 = 20 second event

6) 116-127 Events 116 through 127 are for system resource allocation. Drivers and other utilities requiring ownership of a system resource synchronize on these events. These events are initially set by PDOS, indicating the resource is available. One and only one task at a time is allowed access to the resource. When the task is finished with the resource, it must reset the event thus allowing other tasks to gain access.

116 = Reserved
117 = Reserved
118 = Reserved
119 = Reserved
120 = Level 2 lock
121 = Level 3 lock
122 = Batch event
123 = Spooler event
124 = Reserved
125 = Reserved
126 = Reserved
127 = Virtual ports

7) 128 Event 128 is local to each task. Unlike other events, it can only be set by a delay primitive (XDEV). It is automatically reset by the scheduling of a task suspended on event 128.

128 = Local to each task

2.1.6 TASK COMMUNICATION

Mailbox communication

Many different methods are available for intertask communication in PDOS. Most involve a mailbox technique where semaphores are used to control message traffic. Specially designed memory areas such as MAIL, COM, and event flags allow high level program communications. PDOS currently maintains 32 message buffers for queued message communications between tasks or console terminals. More sophisticated methods require program arbitrators and message buffers.

Absolute data movement

MEM[adr]=data
MEMW[adr]=data
MEML[adr]=data
MEMP[adr,page]=data

Absolute memory locations are referenced by using the BASIC MEM functions. The MEM function moves byte data; MEMW moves words; MEML moves long words; and MEMP moves 8-byte BASIC variables. MEMP passes data between different memory pages in a mapped environment or to a page external to the current task.

Event flags

127 Event flags

EVENT 30

IF EVF[30]

Event flags are global system memory bits, common to all tasks. They are used in connection with task suspension or other mailbox functions. Events 1 through 63 are for software communication flags. Events 64 through 127 automatically reset when a suspended task is rescheduled. Events 81 through 95 are output events; 96 through 111 are input events; 112 through 115 are timing events; and 116 through 127 are system events. Event 128 is local to each task and cannot be used to communicate between tasks.

Message buffers

32 64-byte buffers

PDOS maintains 32 64-byte message buffers for intertask communication. A message consists of up to 64 bytes plus a destination task number. More than one message may be sent to any task. The messages are retrieved and displayed on the console terminal whenever the destination task issues a PDOS prompt or by executing a Get Task Message primitive (XGTM). The displayed message indicates the source task number. The BASIC verbs SENDM and GETM may also be used to pass data between tasks.

(2.1.6 TASK COMMUNICATION continued)

Message pointers

32 4-byte pointers

PDOS supports shorter message pointer transfers between tasks with the Send Message Pointer (XSMP) and Get Message Pointer (XGMP) primitives. When a pointer is sent, event [destination message slot # + 64] is set. When a message pointer is retrieved, the corresponding event is cleared. These messages are not queued, but are much faster for intertask message passing than the queued 64-byte messages.

Memory Mailbox

Memory Mailbox

The FM monitor command is used to permanently allocate system memory for non-tasking data or program storage. Memory allocated in this way can be used for mailbox buffers as well as handshaking semaphores or assembly programs. (See the >FM monitor command.)

2.1.7 TASK SUSPENSION

Task suspended pending event

Any task can be suspended pending one or two events. Software events (1-127) are system memory bits global to all tasks. Event 128 is local to each task. A suspended task does not receive any CPU cycles until one of the desired events occurs. A task is suspended from BASIC by using the WAIT command, or from an assembly language, C, Fortran, or PASCAL program by the XSUI primitive. A suspended task is indicated in the LIST TASK (LT) command by the event number(s) being listed under the 'Event' heading.

```
x>LT
Task   Prt Tm  Event   Map Size     PC     ...
*0/0   64  2            0   384  00001D08 ...
 1/0   64  2   99       0   20   00001B42 ...
x>
```

Immediate and delayed rescheduling

When one of the events occurs, the task is rescheduled and resumes execution. If the event is set by the XSEF primitive, then an immediate context switch occurs. If a high priority task is waiting for the event, it is immediately rescheduled, overriding any current task (unless locked). If the event is set with a XSEV primitive, then the task begins execution during the normal swapping function of PDOS.

2.1.8 HIGH PRIORITY TASKS

Task Prt Tm Event

A high priority task is defined as a task in the execution list which is exempt from round robin scheduling. This means the task will continue to execute until it suspends itself (due to I/O or if an XSUI command is executed,) or a higher priority task becomes ready. Task priority is listed by the LT (List Task) command under the 'PRT' heading. A task priority can be altered with the 'TP' command.

High priority tasks are useful in writing user interrupt handlers where immediate and fast response is required.

2.2 PDOS FILE MANAGEMENT

File management module

The PDOS file management module supports sequential, random, read only, and shared access to named files on a secondary storage device. These low overhead file primitives use a linked, random access file structure and a logical sector bit map for allocation of secondary storage. No file compaction is ever required. Files are time stamped with date of creation and last update. Default PDOS configurations allow up to 32 files to be open simultaneously; however, PDOS may be configured for up to 127 files. Complete device independence is achieved through read and write logical sector primitives.

Sequential, random, read only, and shared file access

2.2.1 PDOS FILE STORAGE

File, file directory

Disk number

256-byte blocked data transfers

A file is a named string of characters on a secondary storage device. A group of file names is associated together in a file directory. File directories are referenced by a disk number. This number is logically associated with a physical secondary storage device by the read/write sector primitives. All data transfers to and from a disk number are blocked into 256-byte records called sectors.

File directory entry

A file directory entry contains the file name, directory level, the number of sectors allocated, the number of bytes used, a start sector number, and dates of creation and last update. A file is opened for sequential, random, shared random, or read only access. A file type of 'DR' designates the file to be a system I/O driver. A driver consists of up to 252 bytes of position independent binary code. It is loaded into the channel buffer whenever opened. The buffer then becomes an assembly program that is executed when referenced by I/O calls.

(2.2.1 PDOS FILE STORAGE continued)

Sector bit map

A sector bit map is maintained for each disk number. Associated with each sector on the logical disk is a bit which indicates if the sector is allocated or free. Using this bit map, the file manager allocates (sets to 1) and deallocates (sets to 0) sectors when creating, expanding, and deleting files. Bad sectors are permanently allocated. When a file is first defined, one sector is initially allocated to that file and hence, the minimum file size is one sector.

PDOS file slots

Sector buffer and status area

A PDOS file is accessed through an I/O channel called a file slot. Each file slot consists of a 38-byte status area and an associated 256-byte sector buffer. Data movement is always to and from the sector buffer according to a file pointer maintained in the status area. Any reference to data outside the sector buffer requires the buffer to be written to the disk (if it was altered) and the new sector to be read into the buffer. The file manager maintains current file information in the file slot status area such as the file pointer, current sector in memory, END-OF-FILE sector number, buffer in memory flag, and other critical disk parameters required for program-file interaction.

Simultaneously OPENed files: default=32
max=127

8 (default) active buffers

Most-recently-accessed resident buffer allocation

PDOS defaults to 32 files that may be open at a time though it may be configured to allow for up to 127. Keeping all sector buffers resident would require prohibitive amounts of system memory. Therefore, only eight sector buffers are actually memory resident at a time. The file manager allocates these buffers to the most recently accessed file slots. Every time a file slot accesses data within its sector buffer, PDOS checks to see if the sector is currently in memory. If it is, the file slot number is rolled to the top of the most recently accessed queue. If the buffer has been previously rolled out to disk, then the most recently accessed queue is rolled down and the new file slot number is placed on top. The file slot number rolled out the bottom references the fourth last accessed buffer which is then written out to the disk. The resulting free buffer is then allocated to the calling file slot and the former data restored.

Frequent access = fast access

Files requiring frequent access generally have faster access times than those files which are seldom accessed. However, all file slots have regular access to buffer data.

Forward and backward linked sector file storage

PDOS allocates disk storage to files in sector increments. All sectors are both forward and backward linked. This facilitates the allocation and deallocation of sectors as well as random or sequential movement through the file.

(2.2.1 PDOS FILE STORAGE continued)

PDOS files are accessed in either sequential or random access mode. Essentially, the only difference between the two modes is how the End-Of-File pointers are handled when the file is closed. If a file has been altered, sequential mode updates the EOF pointer in the disk file directory according to the current file byte pointer, whereas the random mode only updates the EOF pointer if the file has been extended.

Sequential or random access

Two additional variations of the random access mode allow for shared file and read only file access. A file which has been opened for shared access can be referenced by two or more different tasks at the same time. Only one file slot and one file pointer are used no matter how many tasks open the file. Hence, it is the responsibility of each user task to ensure data integrity by using the lock file or lock process commands. The file must be closed by all tasks when the processing is completed.

Shared random, read only random access

Shared random access

A read only random access to a file is independent of any other access to that file. A new file slot is always allocated when the file is read only opened and a write to the file is not permitted.

Read only random access

2.2.2 FILE NAMES

PDOS file names consist of an alphabetic character (A-Z or a-z) followed by up to seven additional characters. An optional one to three character extension is separated from the file name by a colon (:). Other optional parameters include a semi-colon (;) followed by a file directory level and a slash (/) followed by a disk number. The file directory level is a number ranging from 0 to 255. The disk number ranges from 0 to 255.

Legal file names:

```
FILE
A1234567:890;255/127
PROGRAM/3
FILE2;10
```

A file typed as a system I/O device driver has entry points directly into the channel buffer for OPEN, CLOSE, READ, WRITE, and POSITION commands.

TTO,TTA

If the file name is preceded by a '#', the file is created (if undefined) on all open commands except for read only open. When passing a file name to a system primitive, the character string begins on a byte boundary and is terminated with a null.

Auto define

x>CF TEMP,#TEMP2/5

Special characters such as a period or a space may be used in file names. However, such characters may restrict their access. The command line interpreter uses spaces and periods for parsing a command line.

```
FILEN   DC.B    'FILE1/4',0
```

2.2.3 DIRECTORY LEVELS

Each PDOS disk directory is partitioned into 256 directory levels. Each file resides on a specific level, which facilitates selected directory listings. You might put system commands on level 0, procedure files on level 1, object files on level 10, listing files on level 11, and source files on level 20. Level 255 is global and references all levels.

256 directory levels

A current directory level is maintained and used as the default level in defining a file or listing the directory when no directory level is specified.

```
x>LV
Level=1
x>LV 10
Was 1
```

2.2.4 DISK NUMBERS

A disk number is used to reference a physical secondary storage device and facilitates hardware independence. All data transfers to and from a disk are blocked into 256-byte records called sectors.

```
0>SY 1,0
Was 0
1,0>SY
Disk=1,0
1,0>_
```

The range of disk numbers is from 0 to 255. Several disk numbers may share the same secondary storage device. Each disk can have a maximum of 65280 sectors or 16,711,680 bytes.

A default disk number is assigned to each executing task and stored in the task control block. This disk number is referred to as the system disk and any file name which does not specifically reference a disk number defaults to this parameter.

PDOS supports multiple disk directory searches. Up to four disk devices can be associated with each task. When a file is referenced, each directory is searched (in order) until the file is found.

```
1>SY 1,2,3,4
Was 1
1,2,3,4>_
```

Some utility programs make use of the system disk for temporary file storage. By not specifying the disk parameter, the program becomes device independent and defaults to the current system disk.

When a task is created, the parent task's disk number(s) and directory level are copied into the task control block of the new task.

2.2.5 FILE ATTRIBUTES

Associated with each file is a file attribute. File attributes consist of a file type, storage method, and protection flags. These parameters are maintained in the file directory and used by the PDOS monitor and file manager.

8 defined file types

The file type is used by the PDOS monitor in processing the the file. For instance, a file typed as 'EX' (a PDOS BASIC file), calls the BASIC interpreter which loads the file and begins execution with the first line number. A file typed as 'OB' (a 68000 object module), calls the relocating loader to load the object into memory. If a start address tag is included at the end of the file, the module is immediately executed. Otherwise, the system loads the module, prints "PDOS ERR 62 No Start" and returns to the monitor.

Relocatable object only

The following are legal PDOS file types:

AC - Assign console. A file typed 'AC' specifies to the PDOS monitor that all subsequent requests for console character inputs are intercepted and the character obtained from the assigned file.

Batch processes

BN - Binary file. A 'BN' file type has no significance to PDOS but aids in file classification.

OB - 68000 tag object file. Output from the MASM 68000 assembler is in tagged object form. The tag directs the PDOS monitor to load the file into memory (if there was a startin address tag) and execute the program.

Must be relocatable object

SY - System file. An 'SY' file is generated from an 'OB' file. MC68000 object is condensed into a memory image by the 'SYFILE' utility. The first location of a system file is the program entry address.

Generated from OB file

BX - PDOS BASIC binary file. A BASIC program stored using the 'SAVEB' command is written to a file in pseudo-source token format. Such a file requires less memory than the ASCII LIST format and loads much faster. Subsequent reference to the file name via the PDOS monitor automatically restores the tokens for the BASIC interpreter and begins execution.

SAVEB "FILE"

(2.2.5 FILE ATTRIBUTES continued)

EX - PDOS BASIC file. A BASIC program stored using the 'SAVE' command is written to a file in ASCII or LIST format. Subsequent file reference via the PDOS monitor automatically causes the BASIC interpreter to load the file and begin execution.

SAVE "FILE"

TX - ASCII text file. A 'TX' type classifies a file as one containing ASCII character text.

DR - I/O driver. A 'DR' file type indicates that the file data is an I/O driver program and is executed when referenced. An I/O driver must be copied with the >TF monitor command or MTRANS utility.

A PDOS file is physically stored in contiguous sectors whenever possible. A non-contiguous structure results from file expansions where no contiguous sectors are available. Contiguous files have random access times far superior to non-contiguous files. A contiguous file is indicated in the directory listing by the letter 'C' following the file type.

File protection flags determine which commands are legal when accessing the file. A file can be delete and/or write protected.

File storage method and protection flags are summarized as follows:

C - Contiguous file. A contiguous file is organized on the disk with all sectors logically sequential and ordered. Random access in a contiguous file is much faster than in a non-contiguous file since the forward/backward links are not required for positioning.

Contiguous File

* - Delete protect. A file which has one asterisk as an attribute cannot be deleted from the disk until the attribute is changed.

Delete protect

** - Delete and write protect. A file which has two asterisks as an attribute cannot be deleted nor written to. Hence, READ, POSITION, REWIND, OPEN, and CLOSE are the only legal file operations.

Delete and write protect

+ - File altered. A file which has a plus sign as an attribute has been altered.

File Altered

2.2.6 TIME STAMPING

When PDOS is first initialized, the system prompts for a date and time. These values are then maintained by the system clock and are used for time stamping file updates, assembly listings, and other user defined functions.

```
PDOS/68000 R3.2
ERII, Copyright 1983-86
xxxxx BIOS
DATE=00-???-00 10-DEC-86
TIME=00:00:00 10:30
```

When a file is first created or defined, the current date and time is stored with the disk directory entry. This time stamping appears in the 'DATE CREATED' section of a directory listing. From then on, the creation date and time are not changed.

Date created

When a file has been opened, altered, and then closed, the current date and time are written to the 'LAST UPDATE' section of the disk directory entry. The time stamp indicates when the file was last altered by any user.

Last update

2.2.7 PORTS, UNITS, AND DISKS

The terms ports, units, and disks are often confused and hence are explained again:

Ports — Ports are logical input channels and are referenced by numbers 0 through 15. Associated with each port is an interrupt driven input buffer. The BAUD PORT (BP) command binds a physical UART to a buffer.

Units — A unit is an output gating variable. Each bit of the variable directs character output to a different source. Bit 1 (LSB) is associated with U1P$(A6) output port. Likewise, bit 2 is associated with U2P$(A6) output port. The 'SU' and 'SPOOL' commands bind the other bits to the PDOS file structure.

Disks — A disk is a logical reference to a secondary storage device. Disk numbers range from 0 to 255. Several disk numbers may reference the same physical device. The system BIOS deciphers what the disk number means.

```
 ,----------------------------------------------------.
 |F F F F F F F F|E E E|L|A T|ss|--|aa|ii|bb|cccc|1111 |  (each character represents a byte)
 '----------------------------------------------------'
  0                    11 12  14 16 18 20 22  24   28
```

```
F = File Name                    8 characters
E = File Extension               3 characters
L = Directory Level              0-255
A = File Attribute               $80 = AC - Procedure file
                                 $40 = BN - Binary file
                                 $20 = OB - 68000 object module
                                 $10 = SY - System module
                                 $08 = BX - BASIC Token file
                                 $04 = EX - BASIC ASCII File
                                 $02 = TX - ASCII text file
                                 $01 = DR - Driver
T = File Type                    $80 = +  - File altered
                                 $04 = /C - Contiguous file
                                 $02 = /* - Delete protect
                                 $01 = /** - Write protect
s = Start Sector Number          Logical start sector
a = Sectors Allocated to File    Sectors allocated
i = Sector Index of EOF          Sectors used
b = Bytes in EOF Sector          0-252
c = Date/Time Created            hr*256+sc, (yr*16+mn)*32+dy
l = Date File Last Changed          "               "
```

FIGURE 2.4 DIRECTORY FORMAT

2.3 PDOS BIOS

The PDOS Basic I/O Subsystem (BIOS) configures the PDOS environment for different types of hardware peripherals. This includes UARTs, mappers, system LEDs, read/write sector primitives, and disk motor control. Other functions of the BIOS include startup parameters such as auto-start, PDOS prompts, default disk, RAM disk size and location, interrupt vector generation and processing, and MAIL array size.

For a list of the current configurable parameters along with their defaults for your system, refer to the MBIOS:SR file or your Installation and Systems Management guide. Additional information on the BIOS may be found in chapter 8 of this manual.

The BIOS is linked with the PDOS kernel and UART module to form an execution module. The monitor and file manager are added to complete a PDOS system.

2.4 PDOS MONITOR

The PDOS monitor is a resident program which handles the most common PDOS commands. After getting a command line, the monitor calls the command line interpreter to parse the line for commands and parameters. A command line is delimited by a [CR]. If a command line is not complete, your task is suspended pending character inputs.

A list of current resident commands can be found in chapter 3 of this manual as well as in Appendix B.

2.4.1 COMMAND LINE INTERPRETER

The PDOS monitor prompts with the current disk numbers followed by a right angle bracket. PDOS then calls the get line (XGLM) primitive. A command line of up to 78 characters is entered. Various control characters are used to edit the input line. These are summarized as follows:

78-character command buffer

```
   [ESC] = Cancel current line
[CTRL-C] = Cancel current line
[CTRL-I] = Enter insert mode
[CTRL-A] = Recall last entered line
[CTRL-L] = Move right 1 character
[CTRL-H] = Move left 1 character
[CTRL-D] = Delete character under cursor
[RUBOUT] = Delete 1 character to the left
```

Input is normally in replace mode. That is, characters are overwritten in under the cursor. A [CTRL-I] changes the input from replace to insert mode. The mode returns to replace mode when a movement control code is entered. The cursor need not be at the end of the line when the [CR] is entered.

A bell signals one of the following: 1) a rubout is entered and the buffer is empty; 2) an attempt has been made to move past the last character; or 3) a control character other than one of the editing characters has been entered.

Bell => Buffer underflow
        Buffer overflow

Numeric parameters are entered as signed decimal, hex, or binary numbers. All numbers are converted to 2's complement 32-bit integers and range from -2,147,483,648 to 2,147,483,647 (hex $80000000 to $7FFFFFFF). Hex numbers are preceded by a dollar sign ($) and binary numbers by the percent sign (%). (Note: Numbers are not checked for overflow. Hence, 4294967295 is equivalent to -1.)

You can enter more than one PDOS command on a line by separating the commands with a period. Command parameters immediately follow the command name and are separated by commas or spaces. Nested parentheses are used to enclose parameters within parameters. When multiple commands appear on the same line, the remainder of the command line is echoed by the monitor as each command is executed.

```
x>LS.SY 1.LS /1.LV

x>CT (CT (MASM PRGM:SR,,LIST,ERR),20,,2),10,,2

x>SP.LV.SY
FREE=180
USED=190/200
x>LV.SY
LEVEL=1
x>SY
SYS DISK=0
x>_
```

When a line is accepted, it is copied to another buffer from which it can be recalled using the [CTRL-A] character. It is again echoed to the console after a carriage return.

2.4.2 PROCEDURE FILES

When a procedure file is in effect (i.e. a file name has been entered which is typed as 'AC'), then characters are drawn from a file rather than the keyboard. When in this mode, parameter substitution is available. Up to nine unique parameters are available along with an error string.

A substitution parameter is designated by the ampersand (&) character followed by a numeric digit. Digits 1 through 9 specify parameters 1 through 9 while digit 0 is replaced with the last error number (LEN$(A6)) converted to a string. '&#' is replaced by the current task #.

If no parameter was specified, both the ampersand and the digit are deleted (null parameter). Parameters are inserted at the point they are encountered.

An ampersand without a digit following is deleted with the exception that a double ampersand is changed to a single ampersand.

Procedure files can be nested two levels deep. Further attempts to nest a procedure result in error 71. The nested procedure may call any other file except an 'AC' type file.

See also monitor commands IF, GT, and EE.

Parameter substitutions:

&0	Last system error (LEN$(A6))
&1..&9	Command line parameters
&#	Current task #
&&	&

```
0>SF DO
MASM &1:SR,#OBJ&#,&2
IF &0.RC
MSYFL OBJ&#,#&1
RC
0>_
```

```
0>DO TEST,LIST
0>MASM TEST:SR,#OBJ0,LIST/2
68K PDOS Assembler R3.2  01-Nov-86
ERII, Copyright 1983-86
SRC=TEST:SR
OBJ=#OBJ0
LST=LIST
ERR=
XRF=
END OF PASS 1
END OF PASS 2
0>IF .RC
0>MSYFL OBJ0,#TEST
68K PDOS SY File Maker Utility 10/10/84
  Source file = OBJ0
  Destination File = #TEST
0>RC
0>_
```

2.4.3 IMPLIED TASKS

If the command line is preceded with the '@' symbol, then a new task is created and given the rest of the command line to execute. The task defaults to 4k bytes of memory and uses the phantom port (port 0). (See 2.3 PDOS BIOS for variable 'B.SZ1' to change task size.) The implied task facilitates creating small tasks such as copying files or deleting files.

If however, the batch processor task is executing, then the command line is sent to the batch task number via the system message buffers and event 122 is set. A new task is not created and you are prompted for another command. The batch processor will be awakened by event 122 and processes the command line for you.

A similar capability exists with the spooler task. If the first command of a command line is a COPY FILE (CF) and the spooler task is executing, then the command line is sent to the spooler task via the system message buffers and event 123 is set. As above, the spooler task is awakened by event 123 and processes the copy file for you.

CHAPTER 3

PDOS MONITOR COMMANDS

The PDOS monitor is a set of resident routines for handling the most common PDOS commands such as defining and deleting files or listing file directories. Commands are passed to the monitor from the Command Line Interpreter (CLI). A list of memory resident commands is searched followed by the disk directory using the command as the file name.

(PDOS MONITOR COMMANDS continued)

3.1 COMMAND LINE EDITING

The PDOS monitor prompts with the current disk search list followed by a right angle bracket. The prompt is defined in the system BIOS module and can be altered to suit your system. (See MBIOS:SR module.)

0,2,4>_

The PDOS get line (XGLM) primitive is used to get a command line of up to 78 characters into the command line buffer (CLP$). Input is normally in replace mode which means an incoming character replaces the character at the cursor. Various control characters are used to edit the input line. These editing control characters are defined in the system BIOS module or MBIOS:SR and can be altered to suit your system. These are summarized as follows:

78-character command buffer

```
    [ESC] = Cancel current line
 [CTRL-C] = Cancel current line
 [CTRL-I] = Enter insert mode
 [CTRL-A] = Recall last entered line
 [CTRL-L] = Move right 1 character
 [CTRL-H] = Move left 1 character
 [CTRL-D] = Delete character under cursor
 [RUBOUT] = Delete 1 character to the left
```

A [CTRL-I] changes input from replace to insert mode. The mode returns to replace mode when any other editing control code is entered. Replace mode overwrites the character under the cursor. Insert mode inserts a character at the cursor position.

Insert and replace mode

In either mode, the cursor need not be at the end of the line when the [CR] is entered. The command line is parsed as it appears on the screen.

When a line is accepted, it is copied to another buffer (MPB$) where it can be recalled by using the [CTRL-A] character. A [CR] and [LF] are output to the console followed by the recalled line. The cursor is positioned at the end of the line. This is a circular buffer and commands will rotate through it as they are recalled.

Recall last line

Numeric parameters are entered as signed decimal, hex, or binary numbers. All numbers are converted to 2's complement 32-bit integers and range from -2,147,483,648 to 2,147,483,647 (hex $80000000 to $7FFFFFFF). Hex numbers are preceded by a dollar sign ($) and binary numbers by the percent sign (%). (Note: Numbers are not checked for overflow. Hence, 4294967295 is equivalent to -1.)

x>CONVERT %1100101,10,$FFE2

A line beginning with an '*' is ignored.

(3.1 COMMAND LINE EDITING continued)

If the assigned console flag (ACI$(A6)) is set, then the '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'. '&#' is replaced with the current task #.

```
x>SF ASM
MASM &1:SR,#OBJ/8
IF &0.RC
MSYFL OBJ/8,#&1
RC
x>ASM MBACK
x>MASM MBACK:SR,#OBJ/8
68K PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=MBACK:SR
OBJ=#OBJ/8
LST=
ERR=
XRF=
END OF PASS 1
END OF PASS 2
x>IF .RC
x>MSYFL OBJ/8,#MBACK
68K PDOS SY File Maker Utility 04/26/84
  Source file = OBJ/8
  Destination File = #MBACK
            Ident = OAMBACK:SR    2  10426841141
   SECTION LENGTH = E000000340
    Entry Address = 00000000
x>RC
```

3.2 PDOS MONITOR COMMANDS

PDOS monitor commands are memory resident and executed by two character codes. These commands are described in detail in this chapter.

3.2.1 AC - ASSIGN CONSOLE REVIEW

Format: AC <file>

The ASSIGN CONSOLE REVIEW command allows part or all of a procedure file to be executed. The <file> parameter selects the file of commands to be reviewed. As each line is read from the file, it is displayed and you are prompted with a No, Yes, or All prompt. An 'N' reply ignores the line and moves to the next. A 'Y' reply passes the line (and only the line) to the monitor for execution. Finally, an 'A' reply passes the line and all subsequent lines in the file to the monitor for execution.

```
x>SF DOE
SY 4,5,10,12
MASM MPDOSD:SR,#MPDOSD:OBJ
MASM MSYRAM:SR,#MSYRAM:OBJ
MASM MPDOSF:SR,#MPDOSF:OBJ
MASM MPDOSK1:SR,#MPDOSK1:OBJ
MASM MPDOSK2:SR,#MPDOSK2:OBJ
MASM MPDOSK3:SR,#MPDOSK3:OBJ
RC
x>AC DOE
x>SY 4,5,10,12? (Y/N/A)N
x>MASM MPDOSD:SR,#MPDOSD:OBJ? (Y/N/A)N
x>MASM MSYRAM:SR,#MSYRAM:OBJ? (Y/N/A)Y
68K PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=MSYRAM:SR
OBJ=#MSYRAM:OBJ;100
LST=
ERR=
XRF=
END OF PASS 1
SYZ. = $00002800
END OF PASS 2
x>_
```

3.2.2 AF - APPEND FILE

Format: AF <file1>,<file2>

```
x>AF PART2/1,PART1
x>AF PART3/1,PART1
x>AF PART4/1,PART1
x>_
```

The APPEND FILE command concatenates two PDOS files. File <file1> is appended onto the end of file <file2>. The file type attribute of <file1> is transferred to <file2>. <file1> is not affected by the operation.

A [CTRL-C] interrupts this function on a sector boundary, closes both files, and returns to the monitor. This action is reported by the message '^C'.

```
x>AF CHAP04,LIST/2^C
x>_
```

The APPEND FILE command uses the PDOS assembly primitive XAPF.

3.2.3 BP - BAUD PORT

```
Format: BP
        BP {-}<port #>
        BP {-}<port #>,<baud rate>
        BP {-}<port #>,<baud rate>,<type>,<UART base addr>
```

```
x>BP 2,1200              Set port #2 to 1200 bau
x>BP 3,1,2,$1F8010       Set port 3 to 9600 baud
x>_                      type 2, and base address
                         $1F8010
```

The BAUD PORT command initializes a PDOS I/O port and binds a physical UART to a character buffer. The command sets the UART character format, receiver and transmitter baud rates, and enables receiver interrupts.

The first parameter <port #> selects the console port and ranges from 1 to 15. This corresponds to the character input buffers defined in PDOS system RAM (SYRAM). If a minus (-) precedes the port number, then the associated port # is stored in the UNIT 2 (U2P$(A6)) variable.

```
<baud rate>  0 = 19200 baud
             1 = 9600 baud
             2 = 4800 baud
             3 = 2400 baud
             4 = 1200 baud
             5 = 600 baud
             6 = 300 baud
             7 = 110 baud
```

```
x>BP -3,9600          Port 3 as UNIT 2 at 9600 bau
```

The receiver and transmitter baud rates are initialized to the same value according to the <baud rate> parameter. The <baud rate> parameter ranges from 0 to 7 or the corresponding baud rates of 19200, 9600, 4800, 2400, 1200, 600, 300, or 110. Either parameter type is acceptable.

The <type> and <UART base addr> are optional and are included when binding a logical port to a different UART. For <type> information, refer to your Installation and Systems Management guide.

The <port #> can also be used to set or reset the port flags. These are bit positions 8 through 15 of the resulting integer value and are defined to the right. It is recommended that the hex format be used when setting these parameters.

```
x>BP $102,1              Set port #2 with ^S^Q

  $100+port = ^S^Q protocol
  $200+port = Pass control characters
  $400+port = DTR protocol
  $800+port = 8-bit character I/O
```

If the BP command has no arguments, then a listing of all currently installed ports is listed to the console. The 'Task' parameter indicates the currently assigned task to that port.

```
x>BP
Port  Type  fwpi8dcs  Base      Rate   Task
#1     1    00000000  FFFFC071  19200  0
#2     1    00000001  FFFFC031  9600
x>BP 4,0,3,$FFFFC401
x>BP 5,0,3,$FFFFC441
x>BP 6,0,3,$FFFFC481
x>BP 7,0,3,$FFFFC4C1
x>BP
Port  Type  fwpi8dcs  Base      Rate   Task
#1     1    00000000  FFFFC071  19200  0
#2     1    00000001  FFFFC031  9600
#4     3    00000000  FFFFC401  19200
#5     3    00000000  FFFFC441  19200
#6     3    00000000  FFFFC481  19200
#7     3    00000000  FFFFC4C1  19200
x>_
```

3.2.4 CF - COPY FILE

Format: CF <file1>,<file2>

The COPY FILE command copies <file1> into <file2>. The original <file2> is destroyed and replaced by <file1>. The file type attribute of <file1> is transferred to <file2>. <file1> is not affected by the operation.

```
x>CF PROGM:SR,PROGM:SR/1
x>CF FILE1,FILE1:BK
x>_
```

A [CTRL-C] interrupts this function on a sector boundary, closes both files, and returns to the monitor. This action is reported by the message '^C'.

```
x>CF CHAP04,LIST/2[CTRL-C]^C
x>_
```

If the spooler task is executing, then the command line is sent to the spooler via the system message buffers and event 123 is set. The spooler task is awakened by event 123 and processes the copy file for you.

The COPY FILE command uses the PDOS assembly primitive XCPY.

3.2.5 CT - CREATE TASK

Format: CT <command>,<size>,<{time*256+}priority>,<port>

The CREATE TASK command places a new task entry in the PDOS task queue and list. Parameters for the new task include a command line, memory size, task time/priority, and an I/O port. The new task number is reported after the task is created.

```
x>CT PRGM,20,40,2        Create 20K byte task,
TASK #1                        40 priority, port #2
```

The <command> parameter is the command line for the new task. The string is passed to the new task via a message buffer and hence cannot exceed 64 characters in length. If the first parameter is omitted, then the task exits to the PDOS monitor. Multiple commands and parameters may be passed by using parentheses.

```
x>CT HELLO,,1,0                  Spawn scheduler
TASK #2
x>CT (MASM PRGM:SR,PRGM),60  Spawn background
TASK #3                          assembler
```

The amount of memory for the new task is given by <size> and is in 1k byte increments (although rounded to the next 2k byte boundary). The system memory bit map is searched for a contiguous block of memory equal to <size>. If the search fails to find a large enough block, then memory is taken from the parent task and allocated to the new task. Default is 32k bytes.

```
x>CT ,10,,3
TASK #4
```

The <{time*256+}priority> parameter specifies the task time and priority. The range of task priority is from 1 to 255 where 255 is the highest priority. The highest priority, ready task always executes. Tasks on the same priority level are scheduled in a round robin fashion. A new task time slice is specified by adding 256 times the value. A zero time slice defaults to four tics.

```
x>CT WATCH,,1
TASK #5
x>CT HIGHP,,-1
TASK #6
```

The <port> parameter assigns to the new task an I/O port. Port 0 is the default and is called the phantom port. On the phantom port, all character outputs and conditional inputs are ignored while requests for character input result in the task aborting with PDOS error 86. More than one task may be assigned to an output port. The input port is a unique assignment and cannot be shared with another task. Input ports are allocated on a first come basis.

```
x>CT ,4,,2
TASK #7
x>CT ,4,,2
TASK #8
```

After a task is created, the spawned task number is reported. This number is used in killing the new task.

(See also 3.2.26 KILL TASK and 3.2.30 LIST TASKS.)

3.2.6 DF - DEFINE FILE

Format: DF <file{;level}{/disk}>
        DF <file{;level}{/disk}>,<sectors>

```
x>DF FILE1;3/1        Define file on level 3, disk 1
x>DF FILE2            Define "FILE2"
x>DF FILE3;10,20      Define contiguous file of
                         length 20*252 or 5040
                         bytes on level 10

x>DF FILE4;10/2,35
x>_
```

The DEFINE FILE command creates a new file in a disk directory. <File> specifies the file name, and if included, {;level} the file directory level and {/disk} the disk directory number. Defaults for the latter two parameters are the current level and disk number.

The <sectors> parameter specifies the number of contiguous sectors to allocate to the file. One initial sector is allocated if the <sectors> parameter is not specified. Only contiguous files can be defined. A contiguous file facilitates random access to the file data since PDOS can directly position to any byte within the file without following sector links.

If a contiguous file is extended past the original allocation length and a non-contiguous sector is appended to the file, then the contiguous file attribute is deleted. Therefore, even though contiguous files can be extended, you should allocate enough sectors when the file is first defined to handle all anticipated data. Otherwise, random file access slows down.

The length of a contiguous file is specified in sectors. Each sector contains 252 bytes or characters of data. The file size is given by the number of sectors times 252. The maximum PDOS file size is limited by the size of the PDOS logical disk.

Bytes = # of sectors x 252

3.2.7 DL - DELETE FILE

Format: DL <file>

```
x>DL FILE1
x>DL FILE2/3
x>_
```

The DELETE FILE command removes from the disk directory the file specified by <file>. All sectors associated with that file are deallocated in the disk's sector bit map and freed for use by other files on the same disk. A file cannot be deleted if it has previously been either delete- or write-protected. These protection flags must be removed with the 'SA' command before the file can be deleted from the disk.

A sector bit map is maintained by PDOS on each disk so that file creation and deletion does not require a disk compaction routine to recover lost disk space. However, frequent file definitions, deletions, and extensions do create small groups of contiguous sectors which tend to fracture files and make the creation of contiguous files impossible. This is remedied by periodically transferring all files to a newly initialized disk which allocates sectors sequentially for each file.

3.2.8 DM - DELETE MULTIPLE FILE

Format: DM <file list>{,A}

The DM command deletes files from a disk directory according to the <file list>. Each file name to be deleted is output to your console along with a '(Y/N/A)' prompt. If you answer the prompt with a 'Y', then the file is deleted. An 'N' answer does not delete the file. If your answer is an 'A', then the file is deleted along with all subsequent files without further prompts.

```
x>DM @:PAS;@
Delete CMAIN:PAS;130/1? (Y/N/A)Y
Delete CROSSA:PAS;130/1? (Y/N/A)N
Delete CROSSB:PAS;130/1? (Y/N/A)A
Delete CROSSC:PAS;130/1
Delete FIND:PAS;130/1
x>_
```

The <file list> is a file mask that is compared against all specified disk directory entries. File names which match are added to a list built in memory. The format for <file list> is as follows:

```
<file list> = {file}{:ext}{;level}{/disk}{/select...}

where   {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
        {:ext} = 1 to 3 characters (:@=all,*=wild)
      {;level} = directory level (;@=all)
       {/disk} = disk number ranging from 0 to 255
     {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                 Change date (/Fdy-mon-yr,/Tdy-mon-yr)
                      or     (/Fmn/dy/yr,/Tmn/dy/yr)
```

Those files containing the attributes '*' (delete protect) and '**' (delete and write protect) must have those attributes removed with the SA command before they can be deleted.

Note that this command does destroy memory in order to build the file list. Hence, the editor or other last used program cannot be re-entered.

The DM command defaults to all levels. As a result, unless you explicitly declare a level in the file list, files on all levels will be affected.

A second parameter has been added to automatically delete all files in the generated file list. If ',A' follows the file this, then no further prompting occurs and all files are deleted.

Delete all files on level 100

```
x>DM 100
```

Delete all files that do not have an extension, on the current level of disk 4

```
x>DM @/4
```

Delete all files with 2-character names beginning with the letter M, any extension, any level on current disk

```
x>DM M*:@;@
```

Delete all files that have not changed since 1985

```
x>DM ;@/T31-DEC-84
```

Delete all files

```
x>DM ;@,A
```

3.2.9 DN - DOWNLOAD FILE

Format: DN <file name>

The DOWNLOAD FILE command outputs the data contained in the file specified by <file name> to the U2P$(A6) port. There is no modification of the binary data in the file as it is passed to the UART routines. This means that tabs are not expanded and that eight-bit as well as seven-bit data can be downloaded.

DN uses the PDOS assembly primitives read only open (XROO) and read block (XRBF) for reading the data from the file. Data is output to the UART routines via the output date (XPDC) primitive. A [CTRL-C] on the main console aborts the command.

```
x>SU 2,3        Set unit 2 to port 3
x>DN FILE1      Send FILE1 out port 3
```

3.2.10 DT - DATE AND TIME

Format: DT

The DT command outputs the current date and time to the user console. These values can be changed by the ID command.

```
x>DT
DATE=12-Feb-86
TIME=09:03:01
x>_
```

3.2.11 EE - ENABLE ECHO

```
Format: EE
        EE <echo flag>
```

The ENABLE ECHO command loads the PDOS echo flag (ECF$(A6)) which controls terminal output. When the most significant bit of ECF$ is set, then all output through console port primitives is disabled. This bit is set when <echo flag> is nonzero. No parameter is equivalent to 'EE 0' or enable echo.

The current ECF$ flag is defined as follows:

```
        ECF$ = ____ __le
                \\\\ \\\\__ No output
                 \\\\ \\\__ LS body list only
                  \\\\ \\__ Reserved
                   \\\\ \__ Reserved
                    \\\\___ Reserved
                     \\\___ Reserved
                      \\___ Reserved
                       \___ Reserved
```

Console echo is again enabled with the ENABLE ECHO command, when the PDOS primitive XLER (load error register) is executed, or when the monitor requests new console commands. Thus 'EE 1' from the console would not disable output.

This command is useful in procedure files for inhibiting irrelevant console output such as temporary procedures or skipped portions of the procedure.

```
x>SF ASM
EE 1
IF &2=OBJ.GT OBJECT
MASM &1:SR,#OBJ/8
IF &0.RC
MSYFL OBJ/8,#&1
RC
OBJECT
MASM &1:SR,#&1
RC
x>_

x>ASM TEST
x>EE 1
x>_

x>ASM TEST1
x>EE 1
1/6u 0/00000000:4AFC    L1 ADDi.L #2,A9
END OF PASS 2  [1 ERROR]
x>IF 303.RC
x>RC
x>_

x>EE 1
x>EE 1.SF TEST:SR/6
x>_
```

3.2.12 ER - LIST ERROR

Format: ER <error#>

The LIST ERROR command displays the PDOS error message associated with <error#>.

Error numbers range as follows:

BASIC errors	1- 49
PDOS errors	50- 99
Disk errors	100-299
MASM errors	300-399
C errors	400-499
QLINK errors	500-599
Pascal errors	600-699

Pascal, BASIC, C, and FORTRAN errors are described in their respective manuals. Appendix A of this manual contains a list of PDOS, MASM, and QLINK errors. Disk error numbers are system-specific and are described in the xxBIOS:SR file and Installation and Systems Management guide.

```
x>ER 5
PDOS ERR 5 Illegal char
x>ER 53
PDOS ERR 53 Not defined
x>ER 100
PDOS ERR 100 Illegal drive
x>_
```

3.2.13 EV - SET/RESET EVENT

Format: EV
EV {-}<event>

PDOS events are set, reset, or listed with the EV command. A positive <event> value sets the event (1), while a negative value resets the event (0). If no parameter follows the command, then all 128 events are listed to your console as 4 32-bit hexadecimal numbers followed by any pending delay events. The first 32 events are shown in the first constant with event 0 being the most significant bit and so forth.

Current PDOS event definitions are as follows:

```
  1-63 = Software events             120 = Level 2 lock
 64-80 = Software resetting events   121 = Level 3 lock
 81-95 = Output port events          122 = Batch event
96-111 = Input port events           123 = Spooler event
   112 = 1/5 second event            124 = Reserved
   113 = 1 second event              125 = Reserved
   114 = 10 second event             126 = Reserved
   115 = 20 second event             127 = Virtual ports
   116 = Reserved                   128+ = Local event
   117 = Reserved                          (128+task#)
   118 = Reserved
   119 = Reserved
```

```
x>EV
 00000000 00000000 00000000 2000FF00
Event=129  Delay=97
x>EV 42
Was 0
x>EV
 00000000 00200000 00000000 2000FF00
Event=129  Delay=61
x>EV -42
Was 1
x>EV
 00000000 00000000 00000000 2000FF00
Event=129  Delay=19
x>_
```

3.2.14 EX - PDOS BASIC

Format: EX

```
x>EX
*READY
BYE
x>_
```

The PDOS BASIC interpreter is entered via the EX command and exited with the BYE command. A PDOS BASIC program is not altered even though BASIC has been exited and re-entered, until another object or BASIC program is executed.

An error 77 results if the BASIC interpreter is not memory resident.

3.2.15 FE - FOR EVERY

```
Format: FE <file list>,<command line>
        FE (<start>,<end>),<command line>
```

The FOR EVERY command generates a set of command strings that are passed to the PDOS monitor through the Input Message Pointer (IMP$(A6)). These strings are stored in upper task memory and EUM$ is reduced during execution of these commands. EUM$ is restored after all commands are executed.

The command string can be most any command or set of commands with the obvious exception of create task or any other command which might tamper with EUM$. This command string can have substitutions as well as carriage returns and even sublists.

If the first parameter begins with an opening parenthesis, then it is assumed that a start and end number follow. A new command string is generated for each number beginning with <start> and ending with <end>.

Otherwise, the first parameter is a <file list> whose format is defined as follows:

```
 <file list> = {file}{:ext}{;level}{/disk}{/select...}

 where   {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
         {:ext} = 1 to 3 characters (:@=all,*=wild)
       {;level} = directory level (;@=all)
        {/disk} = disk number ranging from 0 to 255
      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                  PDOS attribute (/*,/**)
                  Change date (/Fdy-mon-yr,/Tdy-mon-yr)
                        or    (/Fmn/dy/yr,/Tmn/dy/yr)
```

The <command line> substitution parameters are defined as follows:

```
&F = Full file name or count number
&N = File name
&E = Extension
&L = Level
&D = Disk
 \ = Carriage return
 [ = Start sublist
 ] = End sublist
```

```
x>FE @:SR/10 SA &F,TX
x>SA TTA:SR;3/10,TX
x>SA TTO:SR;3/10,TX
x>SA MBACK:SR;12/10,TX
x>SA MDCOMP:SR;12/10,TX
x>SA MDDMAP:SR;12/10,TX
x>SA MDDUMP:SR;12/10,TX

x>FE @:SR;3/10 MASM20 &F,#&N:OBJ/8
x>MASM20 TTA:SR;3/10,#TTA:OBJ/8
68020 PDOS Assembler R3.2 01-Nov-86
ERII, Copyright 1983-86
SRC=TTA:SR;3/10
OBJ=#TTA:OBJ/8
LST=
ERR=
XRF=
END OF PASS 1
END OF PASS 2
x>MASM20 TTO:SR;3/10,#TTO:OBJ/8
68020 PDOS Assembler R3.2 01-Nov-86
ERII, Copyright 1983-86
SRC=TTO:SR;3/10
OBJ=#TTO:OBJ/8
LST=
ERR=
XRF=
END OF PASS 1
END OF PASS 2
x>_

x>FE (4,10) EE 2[LS 3/&F]EE
x>EE 2
 3   BASIC:OBJ/5    OB +  ...
 3   BASIC20:OBJ/5  OB +  ...
 3   BASIC81:OBJ/5  OB +  ...
 3   BASIC:LIB/5          ...
 3   BMAP/5         TX +  ...
 3   MEDIT:OB/6     OB C  ...
 3   TTA/10         DR C  ...
 3   TTA:SR/10      TX C  ...
 3   TTO/10         DR C  ...
 3   TTO:SR/10      TX C  ...
 3   TTS/10         DR C  ...
 3   TTS:SR/10      TX C  ...
 3   DO/10          AC C  ...
x>_
```

(3.2.15 FE - FOR EVERY continued)

Examples:

x>FE @:SR;4 MASM &F,&N:OBJ	Assemble all :SR files into :OBJ files of the same name
x>FE (4,10) EE 2[LS ;@/&F/F1-Jan-86]EE	List all files on disks 4-10 altered in 1986

Note: FE destroys user memory and any program existing in memory before the FE command cannot be reentered.

3.2.16 FM - FREE MEMORY

Format: FM
FM {-}<size>

The FREE MEMORY command drops memory from your current task. If the <size> parameter is positive, then the memory is deallocated and made available to the system for other task usage. If the <size> parameter is negative, then the memory is simply dropped from the current task and is not recoverable except by a special utility.

See Also: MINST - Memory Install Utility

```
x>FM
FREE=0
x>FM 32
ADR=0016800
x>FM
FREE=32
x>GM
x>FM
FREE=0
x>FM -32
ADR=0016800
x>FM
FREE=0
x>_
```

3.2.17 FS - FILE SLOT USAGE

Format: FS

The FILE SLOT USAGE command lists all files currently open along with file slot information. When the first file is opened, it is assigned slot number 32; as successive files are opened, they are assigned file slots in numerical sequence down to 1. (Read Only Open allocates slots in the opposite order, from 1 to 32.) The file slot maintains information such as the current file pointers and sector indexes. This data is defined as follows:

Normally allocated from 32 to 1

Read only allocated from 1 to 32

Slot	File slot #
Name	File name;level/disk #
ST	File status
SM	Current sector in memory
PT	Current file pointer
SI	Sector index of SM
EOF	Sector index/# of bytes in END-OF-FILE sector
TN	Lock Task/Open Task
BF	Buffer pointer
FLGS	Lock flag/# Shared

File status is defined as:

ST =	$8xxx	Sector altered	$xx80	Altered
	$4xxx	File altered	$xx04	Contiguous file
	$1xxx	Driver in channel	$xx02	Delete protect
	$xAxx	Read only access	$xx01	Write protect
	$x6xx	Shared random access		
	$x2xx	Random access		
	$x1xx	Sequential access		

Example:

```
x>FS
Slot Name           ST   SM   PT       SI   EOF     TN   BF       FLGS
 32  LIST;1/2       C100 0D48 0000AF13 0000 0000/D8 0000 0000AE8A 00000000
x>CT (ASM TEST),100.FS
*Task #1
x>FS
Slot Name           ST   SM   PT       SI   EOF     TN   BF       FLGS
 1   ASM;1/21       0A00 0012 0000ABB9 0000 0000/60 0001 0000AB8A 00000000
 32  LIST;1/2       C100 0D4A 0000AF37 0002 0002/BC 0000 0000AE8A 00000000
```

3.2.18 GM - GET MEMORY

Format: GM
GM <size>

The GM command adds memory to the current task. The amount of memory is specified by <size>. If no parameter follows GM, then all of the available memory is added. No error is reported if the memory request cannot be met.

```
x>FM
FREE=32
x>GM 10
x>FM
FREE=22
x>GM
x>FM
FREE=0
x>_
```

x3.2.19 GO - EXECUTE

Format: GO {,arguments...}
        GO <address>{,arguments...}

The GO command executes a program at an absolute memory address or re-enters an existing program in memory. When there is no argument or the argument is zero, execution begins at the last PDOS entry address (EAD$) which is normally found immediately after the task control block.

If an argument is used, then execution begins at the specified <address>.

```
x>MDUMP 100,110
0064-0073  2F9C 595C 2F9C ....
x>GO ,100,110
0064-0073  2F9C 595C 2F9C ....


*******************************************
*       x>GO {<adr>} PROCESSOR
*
GO      MOVEA.L EAD$(A6),A4 ;GET DEFAULT
        XGNP                ;PARAMETER?
          BLS.S @GO002      ;N
        XCDB                ;Y, CONVERT
          BLE.S ERR67       ;ERROR
        MOVEA.L D1,A4       ;USE NEW ADDRESS
*
@GO002  MOVE.L  A4,D1       ;0?
          BNE.S @GO004      ;N
LEA.L   TBE$(A6),A4 ;Y
*
@GO004  JMP     (A4)            ;GOOD LUCK!!!!
```

3.2.20 GT - GOTO

Format: GT <string>

The GOTO command is used in procedure files to selectively process different commands. When the GOTO command is executed, the procedure file is rewound and all command line entries are ignored until a match is found with the <string> parameter and the command line. All preceding command lines to the match, including the matching command line, are ignored. Execution continues with the next line.

The PDOS console echo flag (ECF$) is set to disable all console output until a match is found or the procedure file is exited. It is again restored after the label is found. Labels beginning with an asterisk are recommended since the monitor ignores them.

The example to the right illustrates the use of the GOTO command. Here, the procedure file ASM will assemble a SY file or a OB file depending upon the second parameter.

```
x>SF ASM
IF &2=OBJ.GT *OBJECT
MASM &1:SR,#OBJ/8
IF &0.RC
MSYFL OBJ/8,#&1
RC

*OBJECT
MASM &1:SR,#&1
RC

x>ASM DEMO
x>IF =OBJ.GT *OBJECT
x>MASM DEMO:SR,#OBJ/8
68K PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=DEMO:SR
OBJ=#OBJ/8
LST=
ERR=
XRF=
END OF PASS 1
END OF PASS 2
x>IF .RC
x>MSYFL OBJ/8,#DEMO
68K PDOS SY File Maker Utility 04/26/84
  Source file = OBJ/8
  Destination File = #DEMO
   SECTION LENGTH = E000000244
    Entry Address = 00000000
x>RC
x>_

x>ASM DEMO,OBJ
x>IF OBJ=OBJ.GT *OBJECT
x>GT *OBJECT
x>MASM DEMO:SR,#DEMO
68K PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=DEMO:SR
OBJ=#DEMO
LST=
ERR=
XRF=
END OF PASS 1
END OF PASS 2
x>RC
x>_
```

3.2.21 HE - HELP

```
Format: HE
        HE <parameter>{,<parameter>...}
```

The HELP command provides error number explanations, tutorial guides to PDOS, user command parameter formats or definitions, utility program listings, disk usage instructions, or other textual messages associated with system software. HELP can be executed without destroying a BASIC or user program.

The user can create his own help files for each individual disk. This could include information on how to use the particular application programs on the disk.

PDOS searches through a file named 'HLPTX' for the HELP <parameter>. All lines beginning with a non-blank or control character are matched against the <parameter>. If the <parameter> agrees, then all lines immediately following the keyword line that begin with a blank or control character are printed. This continues until another line with a non-blank first character is encountered. If no match is found, the routine does not print anything and returns.

If the first character of a line is an exclamation point (!), then the line is printed and the help list stops to wait for another character from your console. This allows multiple page help messages to be in the help file.

If the first character of a line is a pound sign (#), then the current help file is closed and a new help file is opened as specified by the file name following the pound sign. Furthermore, a new parameter is read from the user command and used for subsequent searches. This continues indefinitely.

```
x>HE MON GO
  Command: Begin task execution
   Format: GO
           GO <address>{,<arguments>...}
    Notes: If no address, then executes at last
           entry address (EAD$)
```

(Continued on next page. . .)

```
x>HE
 For further help, enter 'HE '
 followed by one of the following:

    MONITOR {monitor command}
    FILE {file help}
    BASIC
    C
    FORTRAN
    PASCAL
x>HE MONITOR
 Current PDOS resident monitor commands:

 AC - Review procedure  GO - Execute
 AF - Append file       GT - Go to label
 BP - Baud port         HE - Help
 CF - Copy file         IA - If altered
 CT - Create task       ID - Init date
 DF - Define file       IF - Conditional
 DL - Delete file       KM - Kill message
 DM - Delete multiple   KT - Kill task
 DN - Download file     LL - List levels
 DT - Display time      LO - Load file
 EE - Enable echo       LS - List directory
 ER - List error        LT - List tasks
 EV - Events            LV - Directory level
 EX - BASIC             MF - Make file
 FE - For every         PB - Debugger
 FM - Free memory       RC - Reset console
 FS - File slots        RD - RAM disk
 GM - Get memory        RN - Rename file

Hit <CR> to continue.....
```

(3.2.21 HE - HELP continued)

```
x>SF HLPTX
LEVEL 0
 This is level 0
HE
#L1

x>SF L1
LEVEL 1
 This is level 1
HE
#L2


x>SF L2
LEVEL 2
 This is level 2
HE
#L3

x>HE
 This is level 0
x>HE HE
 This is level 1
x>HE HE HE
 This is level 2
x>_
```

3.2.22 IA - IF ALTERED

Format: IA <file name>.<command>

The IF ALTERED command tests and clears the altered file bit of the directory entry specified by <file name>. If the file had the alter bit set (indicated in the directory listing by a '+' under type), then execution of the command line continues. Otherwise, the rest of the line is ignored.

This command is useful in assembly procedures to update object modules when many files are involved and only a few may have changed.

```
x>MF #P
*THIS IS A NEW FILE
x>LS
Disk=SY$DSK/x                              ...
Lev  Name:ext        Type      Size  ...
 1   P                 +C       1/1  ...
Files=1              Used=1/1
x>IA P.SF P
x>SF P
THIS IS A NEW FILE
x>IA P.SF P
x>LS
Disk=SY$DSK/x                              ...
Lev  Name:ext        Type      Size  ...
 1   P                  C       1/1  ...
Files=1              Used=1/1
x>_
```

3.2.23 ID - SET SYSTEM DATE/TIME

Format: ID

The SET SYSTEM DATE/TIME command displays the PDOS header and prompts for the date and time. The PDOS header shows the PDOS system type and copyright declaration. The current BIOS configuration is also displayed.

The date can be entered in either a day, ASCII month, year form or numeric month, day, year.

Any delimiter can be used to separate date and time parameters. Pressing [CR] leaves the old date and time.

```
x>ID
PDOS/68000 R3.2
ERII, COPYRIGHT 1983-86
______ BIOS
DATE=00-???-00 11 1 86
TIME=00:00:00 10:30
x>_

x>ID
PDOS/68000 R3.2
ERII, COPYRIGHT 1983-86
______ BIOS
DATE=00-???-00 01-Nov-86
TIME=00:00:00 10:30
x>_
```

3.2.24 IF - IF PROCESSOR

```
Format: IF <string>.<command>
        IF <string>=<string>.<command>
        IF <string>#<string>.<command>
```

The IF processor allows conditional execution during a procedure file. Parameter substitution is active during procedure files and hence at the completion of a process (such as an assembly), the error register (&0) can be checked. If it is nonzero, then the procedure can be aborted.

The condition delimiters are '=' for strings equal, '#' for string not equal. If no delimiter is used, then the true condition is non-blank.

The IF processor could also be used to check for additional parameters from the procedure file header. &1 through &9 have values equal to the parameters 1 through 9 of the procedure command line.

```
IF &1=Q.GT *LINK
MASM S4BIOS:SR,#S4BIOS:OBJ;101
IF &0.RC
MASM S4BIOSU:SR,#S4BIOSU:OBJ;101
IF &0.RC
MASM S4BIOSW:SR/NFU=20,#S4BIOSW:OBJ;101
IF &0.RC

*LINK
QLINK
Z
DEFINE B$SRAM,$03FC
DEFINE S$SRAM,$9800
GROUP 14,15
SECTION 14,$800
IN S4BIOS:OBJ
IN S4BIOSU:OBJ
IN S4BIOSW:OBJ
IN MPDOS:OBJ
IN MSYRAM:OBJ
MAP GFOSU
MAP GFOSU,#EMAP
SY
OUTPUT #EDOS
END
QUIT
RC
```

3.2.25 KM - KILL MESSAGE

Format: KM
        KM <task #>

The KM command removes all task messages associated with <task #> from the message buffers. If no task is specified, then all messages associated with the current task are listed to the console as well as deleted from the message buffers.

See also 3.2.40 SM - SEND MESSAGE.

```
x>SM
TASK #1: REQUEST #1
TASK #1: REQUEST #2
TASK #4: ANOTHER REQUEST
x>KM 1
x>SM
TASK #4: ANOTHER REQUEST
x>_
```

3.2.26 KT - KILL TASK

Format: KT
        KT {-}<task #>

```
x>KT 2
x>_
```

The KILL TASK command removes a task from the task list and returns the task's memory to the free pool for use by other tasks. Only your current task or a task spawned by your task can be killed. (Task 0 can kill any task except itself or a task that is kill protected.)

Each task is assigned a unique task number which is shown by the LIST TASK command. Only the current task (indicated by '*') or those spawned by the current task (indicated by current task number following a "/" character) may be killed. Task #0 is the system task and cannot be killed.

If a minus sign (-) precedes the task number, then the task's memory is not deallocated to the memory bit map. If the task number is zero, then the current task is killed without deallocating memory. If no parameter is given, then the current task is killed and memory is deallocated.

All open files associated with the killed task are closed by the KT command.

x>KT -2	Kill task # w/o freeing memory
x>KT 0	Kill current task w/o freeing memory
x>KT	Kill current task and free memory

3.2.27 LL - LIST LEVEL

Format: LL <file list>

The LIST LEVEL command lists files by directory level according to the <file list> specification. User memory is destroyed by this command.

The format for the <file list> parameter is as follows:

```
 <file list> = {file}{:ext}{;level}{/disk}{/select...}

 where   {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
         {:ext} = 1 to 3 characters (:@=all,*=wild)
       {;level} = directory level (;@=all)
        {/disk} = disk number ranging from 0 to 255
      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                  PDOS attribute (/*,/**)
                  Change date (/Fdy-mon-yr,/Tdy-mon-yr)
                        or    (/Fmn/dy/yr,/Tmn/dy/yr)
```

Examples:

```
x>LL /5                          List all files on disk 5

x>LL ;@/3/F1-Jan-84/T31-Dec-84   List all files on disk 3
                                      that were altered in 1984
```

```
x>LL /4
Lev 0    A, DOSYSTEM, LOCK, PROFILE
         RECORD, SEARCH, START, UNLOCK
Lev 1    DO:ALL, DO:OBJ, DO20:OBJ
         DOB:OBJ, DOB20:OBJ, DOB81:OBJ
         DOE, FCPU21, MASM20, move
Lev 12   MDCOMP:SR, MPDOSN:SR
Lev 13   DOC31, R3
Lev 100  MBIOS:SR, MDUMMY:SR, MPDOS:SR
         MPDOSD:SR, MPDOSF:SR, MPDOSK1:SR
         MPDOSK2:SR, MPDOSK3:SR, MPDOSM:SR
         MPDTCB:SR, MPDTST:SR, MSYRAM:SR
x>_
```

3.2.28 LO - LOAD FILE

Format: LO <file name>
        LO <file name>,<start address>

The LOAD FILE command loads a PDOS object file into memory but does not begin executing it. The file must be typed 'OB' or 'SY'. The starting load address is optionally specified by <start address>. Otherwise it defaults to immediately following the TCB.

This command can be used to debug files, load multiple files or to load programs outside of known PDOS RAM.

The LOAD FILE command uses the XLDF primitive and loads 'SY' files four bytes at a time. As a result, as many as three extra bytes may be loaded.

```
x>LO MASM
x>PB

x>LO PRGM1,$F00000
x>_
```

3.2.29 LS - LIST DIRECTORY

```
Format: LS <file list>
        LS <file list>,<file>
```

```
x>LS {file}{:ext}{;level}{/disk}{/select}
     {@}    {:@}  {;@}    {/0-255}{/AC}
                                   BN
                                   BX
                                   EX
                                   OB
                                   SY
                                   TX
                                   DR
                                   Fdy-mon-yr
                                   Tdy-mon-yr
```

The LIST DIRECTORY command displays a selected list of disk file names including directory level, file name and extension, file type, file size, start sector address, date of creation, and date of last update. Files are selectively listed according to file name, extension, level, disk number, file attribute, or date of last change. The optional <file> parameter can direct the directory list to a PDOS file.

The format of the <file list> is defined as follows:

```
<file list> = {file}{:ext}{;level}{/disk}{/select...}

where:  {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
        {:ext} = 1 to 3 characters (:@=all,*=wild)
      {;level} = directory level (;@=all)
       {/disk} = disk number ranging from 0 to 255
     {/select} = /AC = Assign Console file
                 /BN = Binary file
                 /BX = PDOS BASIC token file
                 /EX = PDOS BASIC file
                 /OB = 68000 PDOS object file
                 /SY = System file
                 /TX = Text file
                 /DR = System I/O driver
                 /*  = Delete protected
                 /** = Delete and write protected
                 /Fdy-mon-yr = selects files with date of
                               last change greater than
                               or equal to 'dy-mon-yr'.
                               /Fmn/dy/yr format can also
                               be used.
                 /Tdy-mon-yr = selects files with date of
                               last change less than or
                               equal to 'dy-mon-yr'.
                               /Tmn/dy/yr format can also
                               be used.
```

In the file list specification, the '@' character indicates all subsequent characters match and the '*' character is a single character wild card. If you enter 'LS 255', PDOS will display all files on the current disk.

Also displayed with each directory listing is the disk name, the number of files stored on the disk and the number of directory entries available. This information is useful in disk maintenance.

(3.2.29 LIST DIRECTORY continued)

If bit #1 ($02) of the echo flag (ECF$) is set, then the LS header is suppressed and the disk number is appended to the file name. Output is also enabled but the old echo flag is restored after the list file operation.

The directory entries are not necessarily in alphabetical order but in the order they are stored in the disk directory. If an alphabetical listing is desired, the MORDIR utility orders the directory or the MDDMAP utility provides additional directory information in alphabetical order.

See also the following utilities:

7.14 MLDIR - DIRECTORY LIST
7.16 MORDIR - ALPHABETIZE PDOS DIRECTORY

Examples:

x>LS	List all files on current level & disk
x>LS 2	List all files on level 2 of current disk
x>LS ;@	List all files on current disk
x>LS ;@/EX/TX/5	List all 'EX' and 'TX' type files on disk 5
x>LS M@:@;1/OB**/4	List all write protected 'OB' files beginning with the letter 'M' on level 1, disk 4
x>LS ****,LIST	List all 4-character files on current level & disk to the PDOS file LIST
x>LS E**:S@;@	List all 3-character files beginning with the letter 'E' and with an extension beginning with 'S', on all levels
x>LS ;@/T1-Jan-85	List all unaltered files since 1984
x>LS ;@/F1-Jan-86	List all files changed after 1985

(3.2.29 LIST DIRECTORY continued)

Examples:

```
8>LS ;@
Disk=SY$DSK/8                          Files=20/32
Lev  Name:ext      Type     Size      Sect    Date created     Last update
 0   MASM          SY C**   76/76     000F   13:14 16-Jan-85 12:24 18-Dec-85
 0   MJEDY         SY C**   25/25     005B   21:09 06-Jun-84 13:20 17-Jun-85
 0   SY$STRT       AC C       8/8     0005   21:04 06-Jun-84 13:17 17-Jun-85
 1   ASM           AC C       1/1     000D   21:09 06-Jun-84 13:39 17-Jun-85
 1   DO            AC C       1/1     000E   12:36 10-Oct-85 12:36 10-Oct-85
 1   PRINT         EX C       1/1     0074   21:09 06-Jun-84 13:40 17-Jun-85
 2   MBACK         SY C       4/4     0075   21:09 06-Jun-84 13:25 17-Jun-85
 2   MINIT         SY C       5/5     007C   16:31 02-May-84 12:06 10-Oct-84
 2   MLDIR         SY C       5/5     0081   16:31 02-May-84 12:06 10-Oct-84
 2   MLEVEL        SY C       3/3     0086   16:31 02-May-84 12:07 10-Oct-84
 3   TTA           DR C**     1/1     0089   16:32 02-May-84 16:32 02-May-84
 3   TTA:SR        TX +C    11/11     008A   11:01 10-Oct-84 16:20 12-Feb-86
 5   HANGMAN       EX C     21/21     00A6   16:31 02-May-84 12:01 01-Aug-84
 5   HANOI         EX C       6/6     00BB   16:31 02-May-84 16:38 05-Jul-84
 6   CLKADJ        EX C*      8/8     00C1   16:31 02-May-84 16:22 05-Jul-84
 6   UPTIME        EX C*      7/7     00C9   16:32 02-May-84 21:53 05-Jul-84
10   B01           OB C       2/2     00D0   16:31 02-May-84 15:27 18-Oct-85
10   B01:SR        TX C       5/5     00D2   16:31 02-May-84 15:27 18-Oct-85
10   B02           OB C       2/2     00D7   16:31 02-May-84 16:14 11-Apr-85
10   B02:SR        TX C       5/5     00D9   16:31 02-May-84 16:13 11-Apr-85
Files=20           Used=197/197
8>LS
Disk=SY$DSK/8                          Files=20/32
Lev  Name:ext      Type     Size      Sect    Date created     Last update
 1   ASM           AC C       1/1     000D   21:09 06-Jun-84 13:39 17-Jun-85
 1   DO            AC C       1/1     000E   12:36 10-Oct-85 12:36 10-Oct-85
 1   PRINT         EX C       1/1     0074   21:09 06-Jun-84 13:40 17-Jun-85
Files=3            Used=3/3
8>LS *L@:@;@
Disk=SY$DSK/8                          Files=20/32
Lev  Name:ext      Type     Size      Sect    Date created     Last update
 2   MLDIR         SY C       5/5     0081   16:31 02-May-84 12:06 10-Oct-84
 2   MLEVEL        SY C       3/3     0086   16:31 02-May-84 12:07 10-Oct-84
 6   CLKADJ        EX C*      8/8     00C1   16:31 02-May-84 16:22 05-Jul-84
Files=3            Used=16/16
```

(3.2.29 LIST DIRECTORY continued)

```
8>LS ;@/T1-1-85
Disk=SY$DSK/8                               Files=20/32
Lev  Name:ext      Type      Size      Sect    Date created    Last update
 2   MINIT         SY C       5/5      007C  16:31 02-May-84 12:06 10-Oct-84
 2   MLDIR         SY C       5/5      0081  16:31 02-May-84 12:06 10-Oct-84
 2   MLEVEL        SY C       3/3      0086  16:31 02-May-84 12:07 10-Oct-84
 3   TTA           DR C**     1/1      0089  16:32 02-May-84 16:32 02-May-84
 5   HANGMAN       EX C      21/21     00A6  16:31 02-May-84 12:01 01-Aug-84
 5   HANOI         EX C       6/6      00BB  16:31 02-May-84 16:38 05-Jul-84
 6   CLKADJ        EX C*      8/8      00C1  16:31 02-May-84 16:22 05-Jul-84
 6   UPTIME        EX C*      7/7      00C9  16:32 02-May-84 21:53 05-Jul-84
Files=8            Used=56/56
8>LS ;@/F1-1-85/T6-1-85
Disk=SY$DSK/8                               Files=20/32
Lev  Name:ext      Type      Size      Sect    Date created    Last update
 10  B02           OB C       2/2      0007  16:31 02-May-84 16:14 11-Apr-85
 10  B02:SR        TX C       5/5      0009  16:31 02-May-84 16:13 11-Apr-85
Files=2            Used=7/7
8>LS ***;@
Disk=SY$DSK/8                               Files=20/32
Lev  Name:ext      Type      Size      Sect    Date created    Last update
 1   ASM           AC C       1/1      000D  21:09 06-Jun-84 13:39 17-Jun-85
 3   TTA           DR C**     1/1      0089  16:32 02-May-84 16:32 02-May-84
 10  B01           OB C       2/2      000D  16:31 02-May-84 15:27 18-Oct-85
 10  B02           OB C       2/2      0007  16:31 02-May-84 16:14 11-Apr-85
Files=4            Used=6/6
8>LS ;@/**
Disk=SY$DSK/8                               Files=20/32
Lev  Name:ext      Type      Size      Sect    Date created    Last update
 0   MASM          SY C**    76/76     000F  13:14 16-Jan-85 12:24 18-Dec-85
 0   MJEDY         SY C**    25/25     005B  21:09 06-Jun-84 13:20 17-Jun-85
 3   TTA           DR C**     1/1      0089  16:32 02-May-84 16:32 02-May-84
Files=3            Used=102/102
8>LS ;@/SY**
Disk=SY$DSK/8                               Files=20/32
Lev  Name:ext      Type      Size      Sect    Date created    Last update
 0   MASM          SY C**    76/76     000F  13:14 16-Jan-85 12:24 18-Dec-85
 0   MJEDY         SY C**    25/25     005B  21:09 06-Jun-84 13:20 17-Jun-85
Files=2            Used=101/101
8>LS ;@/SY
Disk=SY$DSK/8                               Files=20/32
Lev  Name:ext      Type      Size      Sect    Date created    Last update
 0   MASM          SY C**    76/76     000F  13:14 16-Jan-85 12:24 18-Dec-85
 0   MJEDY         SY C**    25/25     005B  21:09 06-Jun-84 13:20 17-Jun-85
 2   MBACK         SY C       4/4      0075  21:09 06-Jun-84 13:25 17-Jun-85
 2   MINIT         SY C       5/5      007C  16:31 02-May-84 12:06 10-Oct-84
 2   MLDIR         SY C       5/5      0081  16:31 02-May-84 12:06 10-Oct-84
 2   MLEVEL        SY C       3/3      0086  16:31 02-May-84 12:07 10-Oct-84
Files=6            Used=118/118
```

3.2.30 LT - LIST TASKS

Format: LT
LT {mode}

The LT command displays all tasks currently in the task list to the console. Task 0 is the system task and is created automatically during system initialization. This task cannot be killed.

Task listing

Your current task is indicated by an '*' preceding the task number. Following the task number is a slash and the parent task number. Subsequent data provides the current status of each task and is defined as follows:

```
*0/0   => current task
 1/0   => spawned task
```

Task	{*=current}Task #/parent task #
Prt	Task priority (1-255)
Tm	Task CPU tics (1 tic=10 ms)
Event	Suspended event(s)
Map	Task map constant
Size	Task size (k bytes)
PC	Program Counter
SR	Status Register
TB	Task control Block
EM	End of memory
I	Input port number
U	Output unit mask
1	Unit 1 port number
2	Unit 2 port number
4	Unit 4 port number
8	Unit 8 port number

A '+' sign following the task priority indicates that the save flag (SVF$) is enabled for that task.

Further task information can be requested by including a numeric parameter (<mode>). Available modes are 1-7 and are defined as follows:

(Continued on next page. . .)

(3.2.30 LT - LIST TASKS continued)

Mode 1 Selects TCB parameters starting with CLP$. The TCB parameters are defined as follows:

```
 TCB=<--1-> <--2-> <--3-> <--4-> <--5-> <--6-> <--7-> <--8->
     <--9-> <-10-> <-11-> <-12-> <-13-> <-14-> <-15-> <-16->

     1=CLP$ Command Line Pointer
     2=BUM$ Beginning of User Memory
     3=EUM$ End-User-Memory
     4=EAD$ Entry Address
     5=IMP$ Assigned Input Message
     6=ACI$ Assigned Console Inputs
     7=LEN$/SFI$ Error Register/Spooling Unit File ID
     8=FLG$/SLV$/FEC$/0 Task Bit Flags/
                        Directory Level/
                        File Expansion Count
     9=CSC$/PSC$ Clear Screen/Position Cursor
    10=SDS$/SDK$ Alternate Disks/System Disk
    11=EXT$ XEXT$ Address
    12=ERR$ XERR$ Address
    13=CMD$/TID$/ECF$/CNT$ Command Line Delimiter/
                           Task ID/
                           Echo Flag/
                           Column Counter
    14=MMF$/PRT$/SPU$/UNT$ Memory Modified Flag/
                           Input Port #/
                           Spooling Unit Mask/
                           Output Unit Mask
    15=U1P$/U2P$/U4P$/U8P$ Unit 1 Port/
                           Unit 2 Port/
                           Unit 4 Port/
                           Unit 8 Port
    16=0/TWO$ Monitor Temps
```

Mode 2 Lists current executing monitor command (MPB$).

Mode 3 Lists both modes 1 and 2.

Mode 4 Outputs current contents of floating point register (FPA$).

Mode 5 Lists modes 1 and 4.

Mode 6 Lists modes 2 and 4.

Mode 7 Lists modes 1, 2, and 4 (all modes).

Examples:

```
x>LT 1
Task  Prt Tm  Event   Map  Size    PC      SR     TB       EM      I U 1 2 4 8
 0/0   64  1  97/-128  0    548  0000EB44 0000 0000D800 00096800  1 1 1 2 0 0
  TCB=0000D903 0000F59C 00096800 0000D000 00000000 00000000 FFFF0000 00010000
      AA009B3D FFFF0A05 00000000 00000000 00000000 03010001 01020000 00000800
```

(Continued on next page. . .)

(3.2.30 LT - LIST TASKS continued)

```
x>CT ,20,,3
*Task #1
x>LT
Task   Prt Tm  Event   Map  Size    PC      SR     TB       EM      I U 1 2 4 8
*0/0   64  2            0   384  00001D08 2004 0000B000 0006B000  1 1 1 0 0 0
 1/0   64  2   99       0   20   00001B42 2000 0006B000 00070000  3 1 3 0 0 0
x>TP 1,50
x>LT
Task   Prt Tm  Event   Map  Size    PC      SR     TB       EM      I U 1 2 4 8
*0/0   64  2            0   384  00001D08 2004 0000B000 0006B000  1 1 1 0 0 0
 1/0   50  2   99       0   20   00001B42 2000 0006B000 00070000  3 1 3 0 0 0
x>LT
Task   Prt Tm  Event   Map  Size    PC      SR     TB       EM      I U 1 2 4 8
 0/0   64  1   97       0   548  00002052 2004 0000D800 00096800  1 1 1 2 0 0
 1/-1  100 1   127      0   16   000E9D4A 2000 000E9800 000ED800  0 1 0 0 0 0
 2/0   64  1   100      0   32   00002052 2004 000E1800 000E9800  4 1 4 2 0 0
*3/0   64  1            0   100  000023D4 2004 000C8800 000E1800  5 3 5 2 0 0
x>LT 2
Task   Prt Tm  Event   Map  Size    PC      SR     TB       EM      I U 1 2 4 8
 0/0   64  1   97       0   548  00002052 2004 0000D800 00096800  1 1 1 2 0 0
  MPB=MEDIT CHAP04/21
 1/-1  100 1   127      0   16   000E9D4A 2000 000E9800 000ED800  0 1 0 0 0 0
  MPB=
 2/0   64  1   100      0   32   00002052 2004 000E1800 000E9800  4 1 4 2 0 0
  MPB=
*3/0   64  1            0   100  00002E06 2000 000C8800 000E1800  5 3 5 2 0 0
  MPB=LT 2
x>LT 7
Task   Prt Tm  Event   Map  Size    PC      SR     TB       EM      I U 1 2 4 8
 0/0   64  1   97       0   548  00002052 2004 0000D800 00096800  1 1 1 2 0 0
  TCB=0000D900 0000F59C 00096800 0000DD00 00000000 00000000 FFFF0000 20010000
      AA009B3D FFFF0A05 00000000 00000000 FF000005 03010001 01020000 00001500
  MPB=MEDIT CHAP04/21
  FPA=00000000 00000000 0000766E
 1/-1  100 1   127      0   16   000E9D4A 2000 000E9800 000ED800  0 1 0 0 0 0
  TCB=000E9910 000EA250 000ED800 000E9D00 00000000 00000000 00000000 00010000
      AA009B3D FFFF0A04 00000000 00000000 00010014 00000001 00000000 00000400
  MPB=
  FPA=00000000 00000000 00000000
 2/0   64  1   100      0   32   00002052 2004 000E1800 000E9800  4 1 4 2 0 0
  TCB=000E1900 000E1D00 000E9800 000E1D00 00000000 00000000 00000000 20010000
      AA009B3D FFFF0A04 00000000 00000000 FF020005 00040001 04020000 00000000
  MPB=
  FPA=00000000 00000000 00000000
*3/0   64  1            0   100  000023D4 2004 000C8800 000E1800  5 3 5 2 0 0
  TCB=000C8905 000C8D00 000E1800 000C8D00 00000000 00000000 00000220 00010000
      AA009B3D FFFF0A04 00000000 00000000 0003002A 00050203 05020000 00000200
  MPB=LT 7
  FPA=00000000 00000000 00000000
x>_
```

3.2.31 LV - DIRECTORY LEVEL

Format: LV
LV <level>

The DIRECTORY LEVEL command displays or sets the current directory level used in directory listings and file definitions.

The DIRECTORY LEVEL command without any argument displays the current directory level. A file defined without a specified directory level is defined on the current level.

If an argument is specified, it is converted to a number and sets the current directory level. The range is from 0 to 255.

The disk directory is soft partitioned into 256 different groups, facilitating file maintenance. A soft partition means that any file is accessible from any current level. It also means that file names must be unique for each disk number (disk directory).

See also 3.2.27 LL - LIST LEVELS.

```
10>LV
Level=1
10>LS
Disk=WDISK #10/10                        ...
Lev  Name:ext      Type      Size  ...
10>DF PAUL
10>LS
Disk=WDISK #10/10                        ...
Lev  Name:ext      Type      Size  ...
 1   PAUL            C        0/1  ...
10>LS 17
Disk=WDISK #10/10                        ...
Lev  Name:ext      Type      Size  ...
 17  LIBGEN        SY C       5/5  ...
 17  LIBGEN:SR     TX C     43/43  ...
10>LS 0
Disk=WDISK #10/10                        ...
Lev  Name:ext      Type      Size  ...
 0   HLPTX         TX C     32/39  ...
 0   SY$STRT       SY C       8/8  ...
 0   MASM          SY C     75/81  ...
 0   MJEDY         SY C     25/25  ...
 0   QLINK         SY C     43/43  ...
10>LS ;@
Disk=WDISK #10/10                        ...
Lev  Name:ext      Type      Size  ...
 0   HLPTX         TX C     32/39  ...
 0   SY$STRT       SY C       8/8  ...
 0   MASM          SY C     75/81  ...
 0   MJEDY         SY C     25/25  ...
 0   QLINK         SY C     43/43  ...
 2   MBACK         SY C       4/4  ...
 2   MBOBJ         SY C       3/3  ...
 2   MCHATLE       SY C       5/5  ...
 3   TTS           DR C       1/1  ...
 3   TTS:SR        TX C       8/8  ...
 5   AMAZING       EX C     25/25  ...
 5   BEAST         EX C     36/36  ...
x>_
```

3.2.32 MF - MAKE FILE

Format: MF <file>

The MF command allows an ASCII file to be created from the user console. The <file> must be previously defined or preceded by a '#'. The normal line editing is permitted but once a return key has been entered, the line is written to the file.

A [CTRL-C] cancels the line without writing it to the file. An [ESC] terminates the process, closes the file, and returns to the PDOS monitor.

The MF command uses the XGLU PDOS primitive and hence, normal editing control characters are available and lines are limited to 78 characters. Control characters other than those used for editing cannot be entered (i.e. this includes a TAB character.)

```
x>MF #DO
MASM &1:SR,#OBJ,&2
IF &0.RC
MSYFL OBJ,#&1
RC[ESC]
x>DO PRGM1
MASM PRGM1:SR,#OBJ,
SRC=PRGM1:SR
OBJ=#OBJ
LST=
ERR=
XRF=
End pass 1
End pass 2
x>IF .RC
MSYFL OBJ,#PRGM1
x>_
```

3.2.33 PB - PDOS DEBUGGER

Format: PB

The PDOS debugger is entered via the PB command or the PDOS assembly primitive XBUG. It is a single task debugger intended to be memory resident and aid in program development by providing memory inspect and change, single instruction tracing, and breakpoints.

The debugger is initialized when the task is created. It will only briefly be explained here.

Once in the debugger, the 'H' command displays the following menu:

```
x>PB
H
A0-7    A-reg               #          Mem IAC
B{#,a}  Lst/def break       #,#        Mem dump
D0-7    D-reg               #,#+       Disassemble
F       68881 regs          #,#,#{WL}  Find B/W/L
{#}G    Go & break          #(0-7      d(Ax)
M       Last dump           #{+-}#     Hex +/-
N#      0=W,1=B,+2=w/o read
O       Offset              ^D         Disassemble
P       PC                  -          Open previous
Q       Exit                LF         Open next
R       Reg dump            +#         # + offset
S       Status
T       Trace               Trace options:
U       Unit                --------------
V       Control IAC         F/R/M      Dump
W{s,e}  Window              G          Go
X       Set breaks & exit   T          Running
Z       Reset
```

1) Inspect and change memory. A memory location is opened (made ready for change) by entering the address followed by a return. Once open, the value can be altered by entering a new hexadecimal number. The location can be closed by a return, minus sign (which immediately opens the previous location), or a line feed (which immediately opens the next location). A [CR] will re-open the address that was last opened unless a 'Z' (reset) command is entered.

An open location can be disassembled with a [CTRL-D]. Memory can be inspected and changed in word (N0) or byte (N1) mode as well as write only (N2 and N3) mode. Write only mode will not read the data when opening a location and displays as 'xx'.

```
1000[CR]: 03FC -
00000FFE: 0000 4[CR]
[CR]00000FFE: 0004 0[CR]
[CR]00000FFE: 0000 [LF]
00001000: 03FC [LF]
00001002: 2C6D [CTRL-D]    MOVE.L  $00B0(A5),A6 [CR]
N1[CR]
[CR]00001002: 2C [LF]
00001003: 6D [CR]
N3[CR]
[CR]00001003: xx [LF]
00001004: xx [CR]
N0[CR]
```

(Continued on next page....)

(3.2.33 PB - PDOS DEBUGGER continued)

2) Register inspect and change. CPU address and data register can be examined and changed by entering an 'A' or 'D' followed by the register number. The method of change and closing protocol is the same as with memory inspect and change.

```
A0=00000000 [CR]
A2=00000000 100[CR]
[CR]A2=00000100 [CR]
A3=0000C562 [CR]
D4=00000000 100[CR]
[CR]D4=00000100 [CR]
```

3) Memory dumps/disassembly. By entering two numbers separated by a comma, memory is displayed to the screen in hexadecimal and ASCII format. If the end memory address is followed by a plus sign, then the memory is disassembled to the screen.

All addresses are evenized before beginning. The last memory dump can be recalled by simply entering the letter 'M'.

```
100,0[CR]
000100/3C00: 0010 4100 0000 0000 ...
M
000100/3C00: 0010 4100 0000 0000 ...
100C,1014+
00100C/4B0C: 1000          MOVE.B  D0,D0
00100E/4B0E: 6718          BEQ.S   $FFFF4B28
001010/4B10: 4E6E          MOVE.L  USP,A6
001012/4B12: 2D0C          MOVE.L  A4,-(A6)
```

4) Register dumps. By entering the character 'R', all 68000 registers are dumped to the screen along with the program counter, supervisor stack pointer, and status register.

```
R
REGISTER DUMP: PC=0000C50A  SP=0000C32E  SR=
D0: 00000020 00000003 00000000 00000000  ...
A0: 00000000 00000000 00000000 00000000  ...
```

5) Base register addressing. By entering a displacement number followed by a left parentheses and an address register number, a displacement address can be opened. The debugger then returns to the memory inspect and change mode and the method of change and closing is the same.

```
A6=0000C000 [CR]
10(6)
0000C010: 0000 [CR]
```

6) Data searching. Memory can be searched for a byte, word, or long word by entering three numbers; namely, start address, end address, and value. The end delimiter determines the type of search. A return does a byte search while a 'W' does a word (16-bit) and an 'L' does a long word (32-bit). As each location is found, the address is displayed. A [CTRL-C] will interrupt this command.

```
2300,2400,1[CR]
  002307  00232A  002373  002389  0023D4
2300,2400,1W
  002372  002388
2300,2400,1L
  002386
```

7) Program counter inspect and change. The program counter is opened by entering the character 'P'. It can be closed with a return or changed by entering a new value followed by a return.

```
P=0000C500 [CR]
P=0000C500 D000[CR]
P=0000D000 [CR]
```

(Continued on next page....)

(3.2.33 PB - PDOS DEBUGGER continued)

8) Status register inspect and change. The status register is opened by entering the character 'S'. It can be closed with a return or changed by entering a new numeric value followed by a return.

```
S=.....0........ 2[CR]
S=.....0......V. [CR]
```

9) Address offset. To facilitate the use of assembly listings, an offset register is provided. Numbers can be entered as a displacement from the offset by preceding them with a plus (+) sign. The offset is inspected and changed by the 'O' character. The offset defaults to $500 beyond the task TCB.

```
O=0000C500 [CR]
O=0000C500 D000[CR]
O=0000D000 C500[CR]
C500,O[CR]
00C500/0000: 6100 179E 4CEE 30E0 ...
+0,O[CR]
00C500/0000: 6100 179E 4CEE 30E0 ...
```

10) Program breaks. The debugger has four break registers. These are set and listed with the 'B' character. If the 'B' is followed by a return, then all current breaks are listed. A 'B' followed by a '0', '1', '2', or '3' and a [CR] will clear the specified break register. If a third parameter follows, then the break is set at that address and the instruction is disassembled to the screen.

```
B1,800[CR]          BRA.L    $FFFF4536
B[CR]
Break #1  000800/4300  BRA.L   $FFFF4536
B1[CR]
B[CR]
```

11) Instruction trace. User programs can be entered for single stepping with the 'T' command. The debugger uses the trace feature of the 68000. Each instruction is disassembled to the screen before it is executed. The absolute instruction address, offset address, value, 68000 assembly mnemonic, and current status register are displayed.

After each instruction is displayed, a [SPACE] will execute it and display the next instruction to be executed. An 'M' character will dump to the screen last memory dump. An 'R' character will dump the current registers, PC, status, and supervisor stack pointer. A 'T' character will put the debugger in a non-stop tracing mode. This is interrupted with any key. A 'G' character will exit the trace mode and continue program execution. And finally, an [ESC] will exit to the debugger command line.

Various anomalies appear when using the trace exception. First, the next instruction after an A-line exception (PDOS call) does not appear but is executed. This is because the 68000 traces after the instruction is executed and the debugger lists instructions before they are executed.

```
P=0000C504 +4[CR]
T
T> 00C504/0004: 7064           MOVEQ.L #$64,D0[SP]
T> 00C506/0006: 4298           CLR.L   (A0)+[SP]
T> 00C508/0008: 5340           SUBQ.W  #1,D0[SP]
T> 00C50A/000A: 6EFA           BGT.S   $00000006[SP]
T> 00C506/0006: 4298           CLR.L   (A0)+[SP]
T> 00C508/0008: 5340           SUBQ.W  #1,D0[SP]
T> 00C50A/000A: 6EFA           BGT.S   $00000006R
REGISTER DUMP: PC=0000C50A  SR=.............
D0: 00000062 00000000 00000000 00000000  ...
A0: 0000C518 00000000 00000000 00000000  ...[ESC]
B1,+C[CR]                      ADD.L   D0,D1
B[CR]
Break #1  00C50C/000C          ADD.L   D0,D1
G
B> 00C50C/000C: D280           ADD.L   D0,D1
T> 00C50E/000E: A00E           Aline   $A00ER
REGISTER DUMP: PC=0000C50E  SR=..........Z..
D0: 00000000 00000000 00000000 00000000  ...
A0: 0000C6A0 00000000 00000000 00000000  ...
```

(Continued on next page....)

(3.2.33 PB - PDOS DEBUGGER continued)

12) Program execution. The 'G' command will either execute a given number of instructions and then break to the monitor or simply continue execution of the user program at the current program counter.	100G G
13) Trace windowing. Program tracing can be windowed by setting trace bounds with the 'W' command. The first parameter specifies the window base and the second specifies the window end address. When the PC is outside the range of the window, the debugger executes the program without displaying instructions or halting at breakpoints. The default window is 0 through $FFFFFFFF.	Z W[CR] Bounds=00000000,FFFFFFFF W 0,F00000[CR] Bounds=00000000,00F00000
14) Output unit selection. The 'U' command is used to direct console output to other PDOS ports.	U3[CR]
15) Reset debugger. The debugger is reset with the 'Z' command. The program counter and offset are set to the task TCB plus $500. All registers including status and breaks are cleared. The debugging window is set to $00000000 through $FFFFFFFF.	Z
16) Debugger mode. Memory can be inspected and changed in word (N0) or byte (N1) mode as well as write only (N2 and N3) mode. Write only mode will not read the data when opening a location and displays as 'xx'.	1000: 03FC [CR] N1[CR] [CR]00001000: 03 [CR] N2[CR] [CR]00001000: xxxx [CR] N3[CR] [CR]00001000: xx [CR] N0[CR]
17) Arithmetic. Hexadecimal numbers can be added or subtracted by separating two numbers with a plus or minus sign.	00001000[CR]: 03FC [CR] 100+456[CR]=00000556 100-457[CR]=FFFFFCA9
18) Exit debugger. There are two ways to exit the debugger. The 'Q' command exits the debugger normally without any adjustment to the object code. The 'X' command sets all current breaks and then exits.	Q x> X x>
19) Control registers IAC. The 68010/20 control registers can be examined with the 'V' command. These include the Destination Function Code (DFC) register, Source Function Code (SFC) register, Cache Address Register (CAAR), and the Cache Control Register (CACR).	V DFC=$00000003 0 SFC=$00000005 CAAR=$55E5BB9F CACR=$00000001

(Continued on next page....)

(3.2.33 PB - PDOS DEBUGGER continued)

20) <u>68881 floating point registers.</u> The 68881 floating point registers are displayed with the 'F' command. The 'F' command can also be used during trace to list the current 68881 register values in extend precision and packed decimal formats. The floating point control register (FPCR), status register (FPSR), and instruction address register (FPIAR) are also displayed. An inspect and change mode is entered for the 68881 control register and status register after the list.

Example:

```
F
FP0.D=3FE8A3D70A3D70A4  FP0.P=4001 00077000000000000002
FP1.D=4059000000000000  FP1.P=0002 000A0000000000000000
FP2.D=7FFFFFFFFFFFFFFF  FP2.P=7FFF 0000FFFFFFFFFFFFFFFF
FP3.D=7FFFFFFFFFFFFFFF  FP3.P=7FFF 0000FFFFFFFFFFFFFFFF
FP4.D=FFFFFFFFFFFFFFFF  FP4.P=FFFF 0000FFFFFFFFFFFFFFFF
FP5.D=FFFFFFFFFFFFFFFF  FP5.P=FFFF 0000FFFFFFFFFFFFFFFF
FP6.D=7FFF0000FFFFFFFF  FP6.P=7FFF FFFFFFFFFFFFFFFFFFFF
FP7.D=7FFF0000FFFFFFFF  FP7.P=FFFF FFFFFFFFFFFFFFFFFFFF
FPCR=00000000  FPSR=00002088  FPIAR=7FFF0000
FPCR=$00000000
FPSR=$00002088
```

In addition, the 68020 PDOS disassembler supports 68020 addressing modes:

```
(bd,zAn,zRi{*s})          ([bd,zPC,zRi{*s}],od)
(bd,zPC,zRi{*s})          ([bd,zAn],zRi{*s},od)
([bd,zAn,zRi{*s}],od)     ([bd,zPC],zRi{*s},od)
```

the new 68020 instructions:

BKPT	PACK	BFCLR	CHK2	CALLM	UNPK
BFSET	MULx.L	CAS2	TRAPcc	BFINS	DIVx.L
CAS	BFTST	BFEXT	CHK	LINK.L	BFCHG
CMP2					

68881 floating point instructions:

FABS	FETOX	FMOVECR	FSIN	FACOS	FETOXM1
FMOVEM	FSINCOS	FADD	FGETEXP	FMUL	FSINH
FASIN	FGETMAN	FNEG	FSQRT	FATAN	FINT
FNOP	FSUB	FATANH	FINTRZ	FREM	FTAN
FBcc	FLOG10	FRESTORE	FTANH	FCMP	FLOG2
FScc	FTENTOX	FCOS	FLOGN	FSAVE	FTRAPcc
FCOSH	FLOGNP1	FSCALE	FTST	FDBcc	FMOD
FSGLDIV	FDIV	FMOVE	FSGLMUL		

and disassembly of PDOS A-line primitives.

(Continued on next page....)

(3.2.33 PB - PDOS DEBUGGER continued)

21) Error exception processing. One of the first clues that something is wrong in a program is the appearance of a message similar to the following:

```
ADR ERR @00B506 30390000 00000123 3031
D0: 0000000A 00000123 0000001C 00000000 00000000 00000005 00000000 00000045
A0: 0000B1B7 0000B1B7 00002304 00000123 00003B3E 00009000 0000B000 0000B3B0
```

This cryptic message tells you that some sort of exception has occurred and provides some details about the state of the program at the time of the program. The first set of letters tell which exception happened. A possible list is shown to the right. For errors other than bus and address errors, only the program counter, the status register, and the instruction is reported.

```
ADR ERR - address error
BUS ERR - bus error
CHCK    - register bounds check
ILLG    - illegal instruction
PRIV    - privileged instruction violation
SPUR    - spurious interrupt
TRCE    - trace trap
TRAP    - TRAP, TRAPcc instructions
OVFLW   - TRAPV instruction
ZDIF    - zero divide
FLIN    - illegal Fxxx instruction
```

Most of these errors occur when the program is trying to access memory illegally or when some part of the code has been overwritten. This can occur especially if stacks overflow into the program code, or if the code is written with non-relocatable instructions which point to undesired locations in memory.

The format of the error message is dependent upon the source of the error. The table to the right describes this format.

ERROR HANDLER FORMATS

ADD,BUS		TRAP,ILL,PRIV,RESV,SPUI,etc.	
	(MESSAGE)		(MESSAGE)
DC.L	PROGRAM COUNTER	DC.L	PROGRAM COUNTER
DC.W	STATUS REGISTER	DC.W	STATUS REGISTER
DC.W	INSTRUCTION REG.	DC.L	0
DC.L	ACCESS ADDRESS	DC.W	INSTRUCTION
DC.W	LADR,R/W,I/N,CODE	DC.W	0

In the above message, the first number following the exception mnemonic is the program counter and is preceded by an '@' sign. The next number is the instruction register and the status register merged into one long word. The third number is the access address. The fourth number is the access state.

The error message shown above describes an address error which occurred when the PC was at $B506. The instruction being executed was $3039 (MOVE.W <adr>,D0). The status register was zero and the invalid address was $00000123. The cause of the error was an attempt to move a word from an odd address.

The debugger will allow you to look at the offending code and help you to evaluate the cause of the difficulty. A problem line like ADR ERR is spotted easily. If your problem is more subtle, you may have to watch it happen. You can enter an XBUG primitive in your code to place you in the trace mode. Or, while you are in the debugger, set break points which will stop execution and return the program to trace mode. When in the trace mode, a 'T' command will allow the trace to proceed until stopped by pressing any key or until the exception error occurs. This will allow you to observe the steps which led to the failure.

3.2.34 RC - RESET CONSOLE

Format: RC

The RESET CONSOLE command is used in an Assigned Console (type=AC) file to terminate the procedure and to revert back to the system console. This allows for a graceful termination of the file commands by closing the file and prompting for a new command.

Since procedure files can be nested, only the current procedure file is closed.

```
x>SF DO          List procedure file
LV.SY
RC
x>SA DO,AC       Set Assigned Console attribute
x>DO             Invoke procedure file
x>LV.SY
LEVEL=1
x>SY
SYS DISK=0
x>RC              Terminates command file
x>_               Waiting for new command
```

3.2.35 RD - RAM DISK

Format: RD
RD {-}<unit>,<size>,<addr>

The RAM DISK command sets or displays the current RAM disk unit, size, and memory address. When the address is changed, the RAM disk must be again initialized. This is easily done by preceding the RAM disk unit by a minus sign. Otherwise, the MINIT utility can be used to initialize the disk.

The Free Memory (FM) command is used to free memory for additional RAM disk memory. The minus sign preceding the size parameter permanently allocates the memory.

```
x>RD                  List current
Disk=8
Size=255
Addr=000ED800
x>FM -578             Free (2560-255)/4=576.25
Addr=0005D000              or 578 (rounded to 2k)
x>RD -8,2560,$5D000
x>SP 8
Files=0/32
Free=2554,2554
Used=0/0
x>_
```

Example:

```
x>LT
Task  Prt Tm  Event  Map  Size   PC       SR   TB       EM       I U 1 2 4 8
*0/0  64  2           0   906   000019CE 2004 0000B000 000ED800  1 1 1 0 0 0
x>RD
Disk=8
Size=255
Addr=000ED800
x>FM -100
Addr=000D4800
x>RD 8,655,$D4800
x>LT
Task  Prt Tm  Event  Map  Size   PC       SR   TB       EM       I U 1 2 4 8
*0/0  64  2           0   806   000019CE 2004 0000B000 000D4800  1 1 1 0 0 0
x>LS /8
Disk=
PDOS ERR 68 Not PDOS disk
x>RD -8,655,$D4800
x>LS /8
Disk=SY$DSK/8                  Files=0/32
Lev  Name:ext     Type     Size     Sect  Date created     Last update
x>SP 8
Free=650,650
Used=0/0
x>_
```

3.2.36 RN - RENAME FILE

```
Format: RN <file1>,<file2>
        RN <file1>,<level>
```

The RENAME FILE command changes the file name stored in the disk file directory. The RENAME command may also be used to move a file from one directory level to another. The file <file1> is renamed to <file2>. A disk specification in the second parameter is meaningless. If a number <level> is used instead of <file2>, the <file1> is moved to the new level.

```
x>RN FILE1,FILE2
x>RN TEMP,PROGRAM2
x>RN PROGRAM2,4
x>RN FILEN/2,FILEN:BK
x>_
```

3.2.37 RS - RESET DISK

Format: RS
RS <disk #>

```
x>RS
x>RS 2
x>_
```

Disk files must be closed at the end of any task so that sector buffers are flushed to the disk, pointers updated in disk directories, and file slots released for further usage. The RESET command either closes all open files associated with your task or closes all open files on a specified disk. The first mode allows your task to terminate itself without affecting the files of other tasks, while the second mode is used before withdrawing a disk from a disk drive.

Assigned console reset

RESET also clears the assigned console FILE ID (ACI$(A6)). However, the assigned console file may not be closed if the RESET disk option is used and the file resides on a different disk.

3.2.38 SA - SET FILE ATTRIBUTES

Format: SA <file>
SA <file>,<attributes>

The SET FILE ATTRIBUTES command associates file attributes with a file in the disk directory. File attributes include file types and protection flags.

Examples:

```
x>SA FILE Clears all attributes (except 'C')
x>SA FILE,OB      Sets OB type only
x>SA FILE,**      Sets protection only
x>SA FILE,OB**    Sets type and protection
```

File types are defined as follows:

AC - Assign console. A file typed 'AC' specifies to the PDOS monitor that all subsequent requests for a console character will be intercepted and the character obtained from the assigned file.

```
x>SA DO,AC        Batch process
```

BN - Binary file. A 'BN' file type has no significance to PDOS but aids in file classification.

```
x>SA OUTPUT,BN    Declare as binary data
```

OB - 68000 object file. All assembly user-defined commands are typed as object files. When the file name is entered at the monitor prompt, PDOS loads the file into memory and executes the program.

```
x>SA SPOOL,OB     Must be relocatable object!
```

SY - System file. A 'SY' file is generated from an 'OB' file. 68000 object is condensed into a smaller and faster loading format by the 'MSYFL' utility.

```
x>SA CONFIG,SY    Must be condensed object!
```

BX - PDOS BASIC binary file. A BASIC program stored using the 'SAVEB' command is written to a file in pseudo-source token format. Such a file requires less memory than the ASCII LIST format and loads much faster. Subsequent reference to the file name via the PDOS monitor automatically restores the tokens for the BASIC interpreter and begins execution.

```
SAVEB "PR:BIN"    File is type "BX"
```

(3.2.38 SET FILE ATTRIBUTES continued)

EX - PDOS BASIC file. A BASIC program stored using the 'SAVE' command is written to a file in ASCII or LIST format. Subsequent file reference via the PDOS monitor automatically causes the BASIC interpreter to load the file and begin execution.	SAVE "PRGM1"	File is typed as "EX"
TX - ASCII text file. A 'TX' type classifies a file as containing ASCII character text.	x>SA LIST,TX	Declare text file
DR - System I/O driver. A 'DR' file type is a PDOS system I/O driver. Channel buffer data is treated as a program and is used to extend the file system to I/O devices.	x>SA PRGM2,DR	

A file can be delete and/or write protected. These parameters follow the file type and are defined as follows:

* - Delete protect. The file is delete protected and cannot be deleted from the disk.	x>SA DATA,*	
** - Delete and write protect. The file cannot be deleted or written to by any PDOS primitive.	x>SA PROGRAM,**	

All file attributes are cleared by omitting the attribute parameter.

All file attributes are cleared by omitting the attribute parameter.	x>SA FOBJECT	Clear all attributes

3.2.39 SF - SHOW FILE

Format: SF {-}<file name>

The SHOW FILE command displays the disk file as specified by <file name> on your console. The output is paged and truncated to 78 characters per line unless the file name is preceded with a minus sign. The output may be temporarily interrupted at any time by striking any key. Output continues when another key is struck. Pressing [ESC] terminates the command at any time.

If a minus sign precedes the file name, then the file is displayed without line truncations or paging. Again, [ESC] terminates the command.

```
x>SF PRGM1                    PRGM1 listed to CRT
 10  INPUT "N=";N
 20  PRINT " FACTORIAL=";FNFACT[N]
 30  GOTO 10
 100  DEFN FNFACT[I]
 110  IF I<=1: FNFACT=1: FNEND
 120  FNFACT=I*FNFACT[I-1]
 130  FNEND
x>_

x>SF -UPTIME
 100  REM UPTIME
 110  DIM D[1],M[2],T[1],W[2]
 120  DATE $D[0]: TIME $T[0]: T=TIC 0
 130  M=$D[0]: D=$D[0;4]: Y=$D[0;7]: C=19
```

Output continues...

3.2.40 SM - SEND MESSAGE

Format: SM
SM <task #>,<message>

The SEND MESSAGE command puts an ASCII text message in a message buffer. The destination is specified by <task#>. The message can be up to 63 characters in length.

If no parameters follow the SM command, then all the current messages in the message buffers are displayed to the console.

Note: No other commands can be appended to an 'SM' command with a period, since the <message> parameter takes everything up to the carriage return.

See also 3.2.25 KM - KILL MESSAGE.

```
x>SM
x>SM 2,HELLO TASK #2
x>SM 2,ARE YOU THERE TASK #2?
x>SM
*Task #2: HELLO TASK #2
*Task #2: ARE YOU THERE TASK #2?
x>SM 0,THIS MESSAGE COMES BACK TO ME!
*Task #0: THIS MESSAGE COMES BACK TO ME!
x>SM
*Task #2: HELLO TASK #2
*Task #2: ARE YOU THERE TASK #2?
x>_
```

3.2.41 SP - DISK SPACE

Format: SP
SP <disk #>

The DISK SPACE command displays the current number of defined files, the total possible directory size, the number of disk sectors free, the largest possible contiguous file, the number of disk sectors used, and the number allocated. All numbers represent decimal sectors. The total size in bytes is the number of sectors times 252.

The <disk #> specifies the disk number. If no parameter is used, then the default disk is displayed.

The 'Files' parameter lists the current number of defined files in the disk directory. This is followed by the maximum number of files definable in the directory.

The 'Free' parameter shows the number of sectors still available for file storage. This is followed by the largest number of contiguous sectors. This is helpful in defining contiguous files.

The 'Used' parameter shows exactly how much of the disk is truly used versus the amount of disk storage allocated. Some files may have END-OF-FILE markers pointing within the file and not at the end. If these files were copied to another disk, the unused storage would be recovered.

```
x>SP
Files=15/128
Free=251,179
Used=7444/7749
x>DF PAUL,180
PDOS ERR 55
x>DF PAUL,179
x>SP
Files=16/128
Free=72,42
Used=7623/7928
x>DL PAUL
x>SP
Files=16/128
Free=251,179
Used=7444/7749
x>SP 6
Files=20/128
Free=39,34
Used=1779/1783
x>_
```

3.2.42 SU - SPOOL UNIT

```
Format: SU
        SU <unit>
        SU <unit>,<file>
        SU <unit>,<port #>
```

The SPOOL UNIT command sets the spool unit and spool file ID variables in the task control block. Whenever the unit and spool unit variables have corresponding bits, then output is directed to the file specified by the spool file ID variable.

```
x>UN 3
Was 1
x>.. Output to main and aux port
x>UN 1
Was 3
x>SU 2,LIST
x>UN 3
Was 1
x>.. Output to main port and file LIST
```

A 'SU 0' closes any open spool file and resets the spool unit variable. An 'RS' command will also close the file but not the spool unit variable.

```
x>SU 0
```

If the second parameter is a number, then it is identified as a port number and is loaded into the output port variables (U1P$(A6), U2P$(A6), U4P$(A6), and U8P$(A6)) according to the unit mask.

```
x>LT
Task  Prt Tm  ... EM       I U 1 2 4 8
*0/0  64  1   ...0ED800    1 1 1 0 0 0
x>SU 2,2
x>LT
Task  Prt Tm  ... EM       I U 1 2 4 8
*0/0  64  1   ...0ED800    1 1 1 2 0 0
x>SU 6,4
x>LT
Task  Prt Tm  ... EM       I U 1 2 4 8
*0/0  64  1   ...0ED800    1 1 1 4 4 0
x>_
```

3.2.43 SV - SAVE TO FILE

Format: SV <file>
SV <file>,<sadr>,<eadr>

The SAVE TO FILE command writes binary memory images to the file specified by <file>. The parameters <sadr> and <eadr> specify the start and end memory bounds. These boundary parameters default to the end of the current TCB (TBE$) and the last loaded address (BUM$).

```
x>SV TEMP,$C000,$D000
```

3.2.44 SY - SYSTEM DISK

Format: SY
SY <disk1>{,<disk2>{,<disk3>{,<disk4>}}}

The disk device identifier is contained within the file name. However, a default or system disks are assigned by the SY command. On all open and define commands, file names without the disk identifier follow the system disk specification order in looking for the file on disk. All other commands use only the first system disk specification.

```
x>SY
Disk=2
2>SY 4,5,2
Was 2
4,5,2>SY
Disk=4,5,2
4,5,2>SY 2
Was 4,5,2
2>SY
Disk=2
2>_
```

3.2.45 TF - TRANSFER FILES

```
Format: TF <file list>,<disk#>
        TF <file list>,<disk#>,A
        TF <file list>,<disk#>,D
        TF <file list>,<disk#>,U
```

The TRANSFER FILE command transfers selected files from one disk unit to another. This command reads as much as possible into memory before writing to the new file and is much faster than the COPY FILE (CF) command. It also retains all file parameters with the exception that unused sectors are not transferred.

Each file name to be transferred is output to your console along with a '(Y/N/A)' prompt. If you answer the prompt with a 'Y', then the file is transferred. A 'N' answer does not transfer the file. If your answer is an 'A', then the file is transferred along with all subsequent files without further prompts.

The <file list> is a file mask that is compared against all specified disk directory entries. File names which match are added to a list built in memory. The format for <file list> follows:

```
 <file list> = {file}{:ext}{;level}{/disk}{/select...}

 where   {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
         {:ext} = 1 to 3 characters (:@=all,*=wild)
       {;level} = directory level (;@=all)
        {/disk} = disk number ranging from 0 to 255
      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                  PDOS attribute (/*,/**)
                  Change date (/Fdy-mon-yr,/Tdy-mon-yr)
                        or    (/Fmn/dy/yr,/Tmn/dy/yr)
```

The optional third parameter allows you to select all files, only defined files, or only undefined files to be transferred. If the parameter 'A' is included, then all transfers will immediately occur. If the parameter is a 'D', then only those files defined on both the source and destination disk units are transferred. If a 'U' parameter is included, then only these files defined on the source disk and NOT defined on the destination disk are transferred. Any errors during these transfers will revert the command back to the prompt mode.

Note that this command does destroy memory in order to build the file list. Hence, the editor or other last used program cannot be re-entered.

```
x>TF 0/10,8
Transfer HLPTX;0/10? (Y/N/A)Y
Transfer SY$STRT;0/10? (Y/N/A)Y
Transfer ASM;0/10? (Y/N/A)Y
Transfer DISKDUP;0/10? (Y/N/A)N
Transfer MASM;0/10? (Y/N/A)N
Transfer MJEDY;0/10? (Y/N/A)A
Transfer PRINT;0/10
Transfer QLINK;0/10
x>TF 0/10,8,A
Transfer HLPTX;0/10
Transfer SY$STRT;0/10
Transfer ASM;0/10
Transfer DISKDUP;0/10
Transfer MASM;0/10
Transfer MJEDY;0/10
Transfer PRINT;0/10
Transfer QLINK;0/10
x>_
```

3.2.46 TM - TRANSPARENT MODE

Format: TM
TM {-}<port #>
TM {-}<port #>,<break>

The TRANSPARENT MODE command directs your current input to <port #>. Input received from <port #> is directed to your output. This command effectively allows you to access other systems as if you were a terminal. ~~If no <port #> parameter is specified, then the current unit 2 port is used.~~

This process continues until an [ESC] character is entered. This can be changed to another character by adding the <break> parameter.

The incoming characters can be stored in memory and later saved to a file by preceding the port number with a minus sign. When the break character is entered, the command will prompt you for a file name and then store all recieved characters in the file.

mup

```
x>TM 2
$ CREATE FILE.PAS        {other system}
[ESC]
x>_

x>TM ,$39                  {break on '9'}
>> 0123456789
x>_                      {return to PDOS}

x>TM -2,2                  {break on ^B}
  <characters saved>
FILE=#DATA
x>_
```

3.2.47 TP - TASK PRIORITY

```
Format: TP <{time*256+}priority>
        TP <task #>,<{time*256+}priority>
```

The TASK PRIORITY command allows you to change task priority of different tasks. The task number is specified by <task #> and priority by <priority>. If only the priority parameter is given, then your current task is affected.

The range of <priority> is 1 to 255, the latter being the highest priority. The highest priority, ready task always executes.

Note: The task time slice can be altered with the TP command by multiplying the new time slice by 256 and adding it to the <priority> parameter.

```
x>TP 0,$440     Time=4, priority=64
```

Example:

```
x>CT (BP -3,0.MASM CPDOSB:SR/RZ=255,,TTA),50
*Task #1
x>LT
Task   Prt Tm  Event   Map  Size   PC       SR   TB       EM       I U 1 2 4 8
*0/0   64  2            0   254   00001D24 2004 0000B000 0004A800 1 1 1 3 0 0
 1/0   64  2            0   50    0000464C 2009 0004A800 00057000 0 1 0 3 0 0
x>TP 1,50
x>LT
Task   Prt Tm  Event   Map  Size   PC       SR   TB       EM       I U 1 2 4 8
*0/0   64  2            0   254   00001D24 2004 0000B000 0004A800 1 1 1 3 0 0
 1/0   50  2            0   50    0004C032 0004 0004A800 00057000 0 1 0 3 0 0
x>_
```

3.2.48 UN - CONSOLE UNIT

Format: UN
UN <unit #>

The CONSOLE UNIT command sets the console output unit number. The unit number selects where the ASCII output is to be directed. Unit 1 is the system console CRT. Unit 2 is the auxiliary output number.

Each bit of the UNIT variable selects a different output device. Various bits can be assigned to different devices or files with the SU command.

```
x>BP -2,9600      Baud port 2 at 9600 for unit
x>UN 3            Output to units 1 and 2 (1+2)

All further ASCII outputs through main port
  and AUX port at 9600 baud.
```

3.2.49 UP - UPLOAD FROM PORT

Format: UP
        UP <port #>
        UP <port #>,<message>

The UPLOAD FROM PORT command loads characters received from port <port #> into user task memory. If no port is specified, then unit 2 port (U2P$) is used. The <message> parameter is first sent out the same port if included. For each 256 characters received, a period is output to the console port. An escape on the user console or from the input stream of characters, a long timeout, or a memory overflow will terminate the command. PDOS then prompts for the file name in which to write the received data.

```
x>UP 2,TYPE FILE.DAT
{COPY FILE $TTO}          Send out port 2
Aux load.........
File=TEMP
x>_
```

3.2.50 ZM - ZERO MEMORY

Format: ZM

x>ZM

The ZERO MEMORY command clears the entire user workspace to zeros. All flags and pointers are reset.

3.3 COMMON PDOS QUESTIONS

The following section deals with some commonly asked questions about PDOS. Although some examples have hardware specific utilities, most all have a direct replacement for your specific system.

Hardware specific utilities have the system ID characters preceding the file name. See your Installation and Systems Management guide for the ID characters for your system.

3.3.1 HOW DO I TRANSFER FILES?

Files can be moved from one disk unit to another with the copy file command (CF), transfer file command (TF), or the disk backup utility (MBACK). Each has its advantages and disadvantages.

The COPY FILE (CF) command is memory resident in the PDOS monitor and moves a file a sector at a time. It does not destroy any existing program in memory but is slower since the heads must move for each sector.

Copy command:

```
x>CF <file1>,<file2>
```

See 3.2.4 CF - COPY FILE.

The TRANSFER FILE command transfers selected files from one disk unit to another. This command reads as much as possible into memory before writing to the new file and is much faster than the COPY FILE (CF) command. It also retains all file parameters with the exception that unused sectors are not transferred.

Transfer command:

```
x>TF <file list>,<disk>{,AUD}
```

Each file name to be transferred is output to your console along with a '(Y/N/A)' prompt. If you answer the prompt with a 'Y', then the file is transferred. A 'N' answer does not transfer the file. If your answer is an 'A', then the file is transferred along with all subsequent files without further prompts.

See 3.2.45 TF - TRANSFER FILES.

The disk backup utility (MBACK) uses memory to buffer large blocks of disk data in backing up a complete disk. It is the fastest method to back up a complete disk. It can be used to back up the disk boot. If you use MBACK to backup a floppy to a larger disk partition, the new disk image will only be floppy-sized. You should only use MBACK to backup floppy disks to floppy-sized partitions. MBACK is not a selective backup. Also, all fractured files remain fractured, all bad sectors remain bad.

See 7.2 MBACK - DISK BACKUP.

Disk backup utility:

```
x>MBACK
68K PDOS Disk Backup Utility
  Source: (Disk # or Disk/Sector) = 0
  Destination: (Disk # or Disk/Sector) = 10
  Number of sectors (# or 'F') = 2528
  Ready?Y
  Backup 'BOOTG..F&.7_..B+'?Y
Reading sector 0...2559
Writing sector 0...2559
SUCCESS!  Disk Name = BOOTG..F&.7_..B+'
x>_
```

3.3.2 HOW DO I USE THE RAM DISK?

The RAM disk is a portion of memory that can be addressed just like a disk. Obviously, a RAM disk provides fast assemblies and disk access.

When the PDOS system first comes up, the RAM disk is defined with 255 sectors and 32 directory entries. If this is sufficient, then go ahead and use it. The procedure file to the right is used to assemble a program and create an 'SY' file using the RAM disk as a scratch pad area.

A more efficient use of the RAM disk might be to copy a floppy disk to it, use it during your terminal session, and then transfer the updated copy back to a floppy disk.

In order to do this, the RAM disk must be made as large as a floppy. This is done as follows:

```
x>FM -578            {578*4+255+1=2568 sectors}
Addr=<address>
x>RD 8,2560,$<address>    {Declare new RAM base}
x>MBACK or MINIT
```

See 3.2.35 RD - RAM DISK and MINIT - INITIALIZE PDOS DISK.

```
x>SF ASM
IF &2=OBJ.GT *OBJECT
MASM &1:SR,#OBJ/8
IF &0.RC
MSYFL OBJ/8,#&1
RC
*OBJECT
MASM &1:SR,#&1
RC
```

```
x>RD
Disk=8
Size=255
Addr=000ED800
x>FM -578
Addr=0005D000
x>RD 8,2560,$5D000
x>RD
Disk=8
Size=2560
Addr=0005D000
x>MBACK
68K PDOS Disk Backup Utility
  Source Disk # = 0
  Destination Disk # = 8
  Number of sectors = 2368
  Ready?Y
  Backup 'DISK #0.........'?Y
Reading sector 1300
Writing sector 1300
Reading sector 2367
Writing sector 2367
SUCCESS!  Disk Name = DISK #0.........'
x>_
```

3.3.3 HOW DO I USE THE EDITOR?

The MEDIT editor is a screen oriented, memory editor. You need only learn a few control characters to quickly become at home using the editor. Here are the basic editor control character commands:

```
    ^G - File retrieve
    ^W - File save

    ^T - Jump to top of text
    ^R - Recenter text
    ^Z - Jump to bottom of file

    ^B - Search backwards for string
    ^F - Search forwards for string

     ^L - Move cursor right
     ^H - Move cursor left
     ^K - Move cursor up
     ^J - Move cursor down

     ^P - Place pointer
[ESC]^P - Position to pointer
     ^U - Buffer fill
[ESC]^U - Buffer insert
```

[ESC]^V - Quit and return to PDOS

The following illustrates the windowing effect used by MEDIT to edit a text file:

```
                           Computer memory
                              ._______.
                    ^T ---> | Text    |         ^D      .____________.
                            |         |      ,-----> |Macro buffer|
                            | ^K v    |     /  ^E     '------------'
                            |     v   |    /
   .________.         ^W   .--+-------+--. /
  /________/| <------------- | Terminal  |      ^U
 | Floppy | | -------------> |  Window   |  ---------> ._________.
 |________|/      ^G   ^R ->|           |  <--------- |Up buffer|
                             |  24 x 80  |   [ESC]^U   '---------'
                             '--+-------+--'
                                |       |
                                |     ^ |
                                | ^J ^  |
                    ^P --->  |.......|         ^B = Search backwards
                                |       |         ^F = Search forwards
                    ^Z --->  '-------'
```

See CHAPTER 5 - PDOS SCREEN EDITOR.

3.3.4 HOW DO I USE PROCEDURE FILES?

Procedure files are very handy in storing a series of PDOS commands for later recall and execution. Such command sequences might be for disk preparation, assemblies, port installation, and other startup procedures.

A procedure file may be created with the MEDIT editor or the MAKE FILE (MF) command. All characters entered in the file will appear as if they had been typed on the keyboard when the procedure file is invoked.

The file type tells the PDOS Command Line Interpreter how to process the file. The 'AC' or Assign Console type declares a file to be a procedure file. The file type is set by the 'SA' command.

See 3.2.38 SA - SET FILE ATTRIBUTES.

Procedure ASM:

```
x>SA ASM,AC
x>SF ASM
IF &2=OBJ.GT *OBJECT
MASM &1:SR,#OBJ/8
IF &0.RC
MSYFL OBJ/8,#&1
RC
*OBJECT
MASM &1:SR,#&1
RC
```

Procedure S4BAUD:

```
x>SF S4BAUD
BP 4,0,3,$FFFFC401
BP 5,0,3,$FFFFC441
BP 6,0,3,$FFFFC481
BP 7,0,3,$FFFFC4C1
BP
RC
```

3.3.5 HOW DO I GET HARDCOPY?

There are many ways to get hardcopy of files or terminal data. Most all involve spooling your unit 2 output to the correct serial or parallel port that is connected to a printer. This is essential for the following to work properly:

1) Use unit 2 output.

```
Example:  x>SU 2,3                {Assign unit 2 as port #3)
          x>UN 3                  {Select units 1 and 2}
          x>SF -LIST              {Print file LIST}
          x>UN 1                  {Turn off unit 2}
```

2) Use PDOS I/O driver.

```
Example:  x>SU 2,3                {Assign unit 2 as port #3)
          x>CF LIST,TTA           {Copy file LIST to TTA}
```

3) Use PDOS file and print later.

```
Example:  x>SU 4,#TEMP            {Assign unit 4 to TEMP file}
          x>UN 5                  {Select units 1 and 4}

          {All output goes to file TEMP and console}

          x>UN 1                  {Select unit 1 again}
          x>SU 0                  {Close TEMP}
          x>@BP -2,1.CF TEMP,TTA  {Baud & copy in background}
          x>_
```

3.3.6 HOW DO I WRITE AN ASSEMBLY PROGRAM?

Assembly programs are very fast and efficient. Using the PDOS system, they are also very easy to write. All console I/O and file management primitives are legal opcodes in the PDOS assembler called MASM or MASM20 (for 68020 systems).

First, use the MEDIT editor to create the assembly program. Be sure to end the program with the 'END' directive and a start address.

Example:

```
*       TEST:SR             05/08/84
*
START   XPMC     MES01     ;ASK FOR A NUMBER
        XGLU               ;GET NUMBER
        XCDB               ;CONVERT TO BINARY
        MULS.W   D1,D1     ;SQUARE NUMBER
        XCBM     MES02     ;CONVERT
        XPLC               ;PRINT RESULT
        XEXT               ;EXIT TO PDOS
*
MES01   DC.B     $0A,$0D,'What is your number? ',0
MES02   DC.B     '  Squared = ',0
        EVEN
        END      START
```

Next, use the MASM assembler to assemble the program.

Example:

```
x>MASM TEST:SR,#TEST
68K PDOS Assembler 3.2
ERII, Copyright 1983-86
SRC=TEST:SR
OBJ=#TEST
LST=
ERR=
XRF=
XRF=
END OF PASS 1
END OF PASS 2
```

Finally, if there were no errors, the program can be executed simply by entering the object module name.

Example:

```
x>TEST
What is your number? 12  Squared = 144
x>_
```

3.3.7 HOW DO I SET UP VIRTUAL PORTS?

PDOS virtual ports (also referred to as "windows") allow selective switching of physical I/O ports to logical task ports. This means that a single terminal can dynamically switch between I/O ports that may be assigned to different tasks or updated by a single task with multiple screen output. Further, a screen image is maintained for all windowed ports so that the switching process also updates the terminal with the current display for that port.

Previously, several terminals had to be used or the user had to write application code to update a single terminal with information according to which screen was selected. This process involved flags and locks or a dispatch task that handled all I/O. With PDOS windows, the system acts as if there were more terminals on the system; multiple tasks are accessible from one terminal.

A high priority window task maintains the virtual screen buffers and handles screen refreshing and buffer printing. A special key sequence is used to switch from one virtual port to another. When a selection is made, PDOS maps your keyboard to another port and the window task clears and updates your display to reflect the current screen.

```
                          Port #1  <-->  Task #0
                         /
                        /
                       /
User Terminal--<---Port #2  <-->  Task #1
                       \
                        \ ....
                         \
                          Port #n
```

Virtual ports allow you to:

1) Easily manage multiple screens.
2) Monitor many different processes.
3) More effectively multi-task with one terminal.
4) Debug screen-intensive programs.
5) Print screen images.

(3.3.7 HOW DO I SET UP VIRTUAL PORTS? continued)

The virtual port process is set up by creating a task with the WIND1 program. The size of the task is equal to the number of ports times two plus four. No I/O port should be assigned.

For example, to generate 15 virtual ports, execute:

```
x>CT (WIND1 {..parameters..}),34
```

If WIND1 encounters an error during its initialization, it will notify its parent task with the appropriate message through the message buffers. Possible errors include:

1. Not enough memory allocated.
2. Window process already executing.
3. Illegal parameters specified.

Next, the program WIND1 signals PDOS that virtual porting is now active by setting the SYRAM variables WIND. and WADR., and allocates buffers for the virtual screens. Further, WIND1 sets its execution priority to 100 and kill-protects itself by setting its parent task to -1. All interactive tasks should have a priority <100. Finally, it suspends on event 127.

The four parameters for WIND1 are as follows:

```
CT (WIND1,<window list>,<port list>,<print>,<append>)
```

where

```
<window list> = LOGICAL WINDOWS (Default=1-15)
  <port list> = PHYSICAL PORTS ALLOWED TO WINDOW (Default=1)
      <print> = ^P OUTPUT FILE OR PORT # (Default=none)
     <append> = ^P APPEND OUTPUT FILE (Default=none)
```

The <window list> parameter specifies the PDOS I/O ports that are to be windowed. The ports are specified by number and are separated by slashes (/). Consecutive ports can be specified by separating the first and last port number with a hyphen (-). Default is 1-15 or all PDOS ports.

For example:

```
1/2/3/8/13/14/15 = 7 ports 1, 2, 3, 8, 13, 14, 15
     1-3/8/13-15 = Same as above
```

(3.3.7 HOW DO I SET UP VIRTUAL PORTS? continued)

The <port list> parameter selects those PDOS I/O ports that are permitted to window. The allows some system security for selected ports. The format is the same as the <window list> and the default is for port 1 only.

The third parameter <print> specifies where a screen dump is sent to. It may be to a file or an I/O port. Whenever the screen dump function is activated ([CTRL-X]P), then the WIND1 program opens the <print> file, outputs the current screen image, and then closes the file. A dump header with the current time and date precedes the output. If a file is used, it must be pre-defined. If an auto-define symbol (#) precedes the filename, the file will be created when necessary.

The forth parameter <append> is similar to the <print> parameter with the following exceptions: 1) only a file can be used for output, and 2) the output is appended to the file. The file must be pre-defined. If an auto-define symbol (#) precedes the filename, the file will be created when necessary.

Examples:

x>CT (WIND1 1/3-5,,2),12

> Creates window processing for ports 1, 3, 4, and 5. Only port 1 is allowed to window and a [CTRL-X]P sends a screen image to port #2.

x>CT (WIND1 1-15,1-4,PBUF,ABUF),34

> Creates windows for all 15 PDOS ports. Physical ports 1 through 4 can window. A [CTRL-X]P sends a screen image to file PBUF and appends the same image onto file ABUF.

```
(WIND.).W = FRPM ____ D__p pppp
            \\\\ \\\\ \\\\    \_
             \\\\ \\\\ \\\\_____ 0-4=PORT #
              \\\\ \\\\ \\\_____ 5  = Reserved
               \\\\ \\\\ \\_____ 6  = Reserved
                \\\\ \\\\ \_____ 7  = WINDOWING DISABLE
                 \\\\ \\\\______ 8  = Reserved
                  \\\\ \\\______ 9  = Reserved
                   \\\\ \\______ 10 = Reserved
                    \\\\ \______ 11 = Reserved
                     \\\\_______ 12 = ALREADY DEFINED
                      \\\_______ 13 = PRINT FLAG
                       \\_______ 14 = REFRESH FLAG
                        \_______ 15 = LEAD FLAG
```

(3.3.7 HOW DO I SET UP VIRTUAL PORTS? continued)

The task is awakened by event 127. When event 127 is set, WIND1 checks the WIND. table for various control bits.

If the refresh bit is set (bit #14), then the PDOS I/O port corresponding to the index number in the table is sent a clear screen command and the current contents of the screen pointed to by the WADR. table.

If the print bit is set (bit #13), then the PDOS I/O port corresponding to the index number in the table is sent a clear screen command and a message indicating a screen dump is occurring. The contents of the corresponding screen are then sent to the print and append files (if specified). Finally, another clear screen command is output followed by the current contents of the screen.

VIRTUAL PORT SELECTION

Virtual ports are selected by a leading control character followed by the port number. (Ports 10 through 15 are selected by letters A through F.) The default control code is [CTRL-X] which is also the clear buffer code. This is alterable at sysgen time by setting B.WND for MBIOS:SR.

A [CTRL-X]P sets the print bit (#13). Two consecutive [CTRL-X]s translate to a single [CTRL-X] which is passed through to the input character processor.

The port number external to PDOS is referred to as the physical. The port number after window translation is referred to as the logical. It is important to understand just where various character control functions happen and which port number is used. These are summarized below:

INPUT		OUTPUT	
PHYSICAL	LOGICAL	LOGICAL	PHYSICAL
--------	-------	-------	--------
^S^Q			^S^Q
H/L Water			H/L Water
8-Bit Mask			8-Bit Mask
	Cntrl Check		
	^C,^C,ESC		
	Get Character	Put Character	
	Set Event	Wait on Event	

(3.3.7 HOW DO I SET UP VIRTUAL PORTS? continued)

CURRENT RESTRICTIONS:

1. PDOS output primitives XPCR and XPDC do not update the row/column counters which are used to store characters in the virtual screen buffers. Hence, they bypass being saved but the output port is translated using the WIND. table.

2. Special screen control functions such as underline, field protect, blink, etc. are not supported by windows. Any screen control characters/sequences not produced by PDOS (not through >TM) are not supported.

3. Position cursor and clear screen control codes are not indivisible when output to the screen. Hence, if a new screen is selected during a position command, then the refresh clear screen command may be out of sync and not work. Simply refresh the screen again.

DISABLING VIRTUAL PORTS (WKILL)

Since the virtual port processor itself (WIND1) contains the screen image buffers, simply killing the task would free memory to PDOS that would still be written to by the character interrupt processor. Hence, the WKILL utility is included to disable virtual port processing.

The WKILL utility allows the virtual port processor to exit cleanly, disabling virtual port processing and resetting crucial pointers.

The format for WKILL is:

    WKILL {<task #>}

The optional parameter <task #> selects the virtual port processor task. WKILL clears the SYRAM variables WIND. and WADR. and unprotects the virtual port processor. Then, a KT <task #> follows.

Note: WKILL can only be executed from task 0.

(3.3.7 HOW DO I SET UP VIRTUAL PORTS? continued)

DIFFERENT TERMINAL TYPES (WTERM)

The virtual port processor initializes its port position cursor and clear screen codes to those of the parent task. Hence, refresh uses the same codes for all ports unless this is altered by the WTERM utiltity after the window process is executing. These codes are located immediately following the address table (WADR.).

The WTERM utility has identical parameter definitions as the PDOS MTERM utility with the exception that the first parameter is a windowing port number. WTERM does not permit parameter passing in user-defined modes. (See the description of the MTERM utility.)

Example:

```
x>WTERM 5,S
x>WTERM
68K PDOS Change Terminal Type Utility
Terminals:
    A=ADDS Regent 25
    D=Decscope (VT52)
    H=Hazeltine 1520
    I=Intertube II
    L=Lear Seigler ADM3a
    S=Soroc IQ120
    M=Data Media Excel 12
    V=VT100 / ANSI terminal
    U=User Defined
Port #=6
Type = V
x>
```

(3.3.7 HOW DO I SET UP VIRTUAL PORTS? continued)

VIRTUAL PORT PROCESS MONITOR (WLOOK)

The virtual port monitor utility WLOOK displays the screen buffer addresses, the current refresh clear screen/position cursor codes, and then dynamically displays the current window translation table (WIND.).

Example:

```
x>CT (WIND1 1/3-6,1/4),16
*TASK #1
x>WLOOK
WINDOW BUFFERS:

 #1=$000EA23C  #2=Undefined  #3=$000EA9BC  #4=$000EB13C  #5=$000EB8BC
 #6=$000EC03C  #7=Undefined  #8=Undefined  #9=Undefined  #A=Undefined
 #B=Undefined  #C=Undefined  #D=Undefined  #E=Undefined  #F=Undefined

PORT CLEAR/POSITION CODES:

 #1=$AA009B3D  #2=$AA009B3D  #3=$AA009B3D  #4=$AA009B3D  #5=$AA009B3D
 #6=$AA009B3D  #7=$AA009B3D  #8=$AA009B3D  #9=$AA009B3D  #A=$AA009B3D
 #B=$AA009B3D  #C=$AA009B3D  #D=$AA009B3D  #E=$AA009B3D  #F=$AA009B3D

Enter [ESC] to exit to PDOS

 0006 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 8006 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 0001 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 8001 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 0006 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
```

For more information on how to use virtual ports, consult the PDOS utilities WIND1, WKILL, WTERM, and WLOOK (in chapter 6 of this manual). The internals of the virtual porting process are described in Appendix H of this manual.

CHAPTER 4

PDOS ASSEMBLY PRIMITIVES

PDOS assembly primitives are assembly language system calls to PDOS. They consist of one word A-line instructions (words with the first four bits equal to hexadecimal 'A'). PDOS calls return results in the 68000 status register as well as regular user registers.

PDOS calls are divided into three categories: namely, 1) system, 2) console I/O, and 3) file support primitives.

(TABLE OF CONTENTS continued)

(TABLE OF CONTENTS continued)

4.4 SYSTEM CALLS

(TABLE OF CONTENTS continued)

(TABLE OF CONTENTS continued)

4.5 CONSOLE I/O PRIMITIVES

(TABLE OF CONTENTS continued)

4.6 FILE PRIMITIVES

4.1 GUIDELINES FOR 68000 ASSEMBLY PROGRAMMING

The following guidelines are to be used in assembly programming for the PDOS system:

1) Standard 68000 Assembly Language.

The PDOS assembler supports the standard Motorola 68000 assembly language instruction set as defined in the M68000 16-/32-bit Microprocessor Programmer's Reference Manual. This includes register designations, instruction mnemonics, and addressing syntax.

Standard 68000 assembly support

2) 68000 Register Usage.

All 68000 registers are available for user programs. However, as a convention, the following are recommended register usages:

XGML reloads A5 & A6

```
A4 = User variables base register
A5 = SYRAM pointer (initialized by PDOS)
A6 = TCB pointer (initialized by PDOS)
A7 = User stack pointer (EUM$-$100).
```

3) Position Independent and Re-entrant Coding.

PDOS assembly programs should be position independent and re-entrant coded. This means that base registers and PC relative variables should be used in the place of absolute addressing and that the stack or registers should be used for parameter passing.

For example:

```
GOOD                              BAD
----------------------------      --------------------------

      BSR.L   SUBRT                     JSR      SUBRT         Use BSR's instead of JSR's


      LEA.L   LAB(PC),A0                MOVEA.L #LAB,A0        Use (PC) instead of absolute
      ...                               ...
LAB   EQU     *                   LAB   EQU      *


      LEA.L   VARS(PC),A0               CLR.B    PRT           Setup OFFSET area
      CLR.B   PRT_(A0)                  ...
      ...                               PRT      DC.B    0
VARS  EQU     *
      OFFSET  0
PRT_  DS.B    1
```

(4.1 GUIDELINES FOR 68000 ASSEMBLY PROGRAMMING continued)

4) PDOS Primitives.

PDOS assembly primitives are fully supported by the PDOS assembler. These calls to PDOS will assemble to A-line instructions.

```
XEXT
XSOP
```

5) System Variables.

The PDOS assembler supplies most system constants required by a user. These constants are supplied on reference after the 'OPT PDOS' directive is executed. The following is the standard convention adopted for external PDOS symbols:

```
 xxx$ = TCB index (A6)            MOVE.B  U1P$(A6),D0
 xxx. = SYRAM constant            MULU.W  #TBZ.,D0
xxxx. = SYRAM index (A5)          MOVE.L  TICS.(A5),D1
 .xxx = Global system constant    MOVE.W  #.BPS,D7
m.xxx = Module constant           MOVE.W  #B.PTMSK,SR
m$xxx = Module entry point        BSR.L   K2$PINT
m_xxx = Module index              CLR.W   B_TPS(A0)
 xxx_ = User index                ADDA.L  AVL_(A4),A0
```

The following illustrates how some of these might be used:

```
BSET.B #~118,118/8+EVTB.(A5)      Set event 118

MOVEA.L MAIL.(A5),A0              Point to the MAIL array

MOVE.L  TICS.(A5),D1              Read system tics

ST.B    DFLG.(A5)                 Set hard partitioned directory

ST.B    TLCK.(A5)                 Lock current task

MOVE.B  #2,PRT$(A6)               Set input port #

MOVE.B  #5,FEC$(A6)               Set file expansion count

ST.B    ECF$(A6)                  Disable console echo

MOVEA.L BIOS.(A5),A0              Read system ID characters
MOVE.W  B_SID(A0),D0
```

(4.1 GUIDELINES FOR 68000 ASSEMBLY PROGRAMMING continued)

6) Assembly Format.

PDOS assembly text has the following conventions:

a. A comment line before any entry address.
b. 2 spaces preceding a conditional branch.
c. Semi-colon with space for comment.

```
*
LABEL   CMPI.W  #10,D1   ; LESS THAN 10?
          BLT.S LABEL    ; Y
```

7) Source file documentation.

PDOS source files have the following conventions:

```
        TTL     FILE - PDOS PROGRAM FILE          Assembler TTL directive
*       FILE:SR   07/22/85                        File name followed by last update date
*****************************************
*                                       *
*       FFFFFF IIII LL     EEEEEE       *
*       FF      II  LL     EE           *
*       FF      II  LL     EE           *
*       FFFFF   II  LL     EEEEE        *
*       FF      II  LL     EE           *
*       FF      II  LL     EE           *
*       FF     IIII LLLLLL EEEEEE       *
*                                       *
*=***************************************
*       Eyring Research Institute Inc.            Company identification
*       Copyright 1983-86                           with copyright notices
*       Proprietary Software
*       ALL RIGHTS RESERVED
*=
*=             Module Name: FILE                  Module identification
*=                  Author: John Doe              Author of program
*=      Changes Authorized by:                    Who authorizes any changes
*=          Revision History:
*=
*=      DATE     R.V    DESCRIPTION
*=
*=      07/08/85 2.36   D$INT called from XCTB    Revision history
*=      07/18/85 2.37   XLER enables echo ECF$
*=
FILE    IDNT    2.37    M68000 PDOS               Program ID
*=
*=***************************************
        PAGE
```

4.2 PDOS ASSEMBLY LANGUAGE CALLS

PDOS assembly primitives are one word A-line instructions which use the exception vector at memory location $00000028. Most primitives use 68000 registers to pass parameters to and results from resident PDOS routines.

Trapping an error after a PDOS call:

```
CALLX  LEA.L   FILEN(PC),A1 ;GET FILE NAME
       XSOP                 ;OPEN FILE, ERROR?
         BNE.S ERROR        ;Y
       MOVE.W  D1,SLTN(A4)  ;N, SAVE SLOT #
```

PDOS primitives return error conditions in the processor status register. This facilitates error processing by allowing your program to do long or short branches on different error conditions.

PDOS command primitives can be grouped into six levels according to their function and calling hierarchy. These levels are System Calls, System Support Calls, Console I/O Calls, File Support Calls, File Management Calls, and Disk Access Calls.

Level 1 PDOS primitives consist of system calls that deal with functions such as swapping, message passing, events, TRAP vector initialization, etc. The PDOS system calls are as follows:

System Calls

XGML - Get memory limits
XGUM - Get user memory
XFUM - Free user memory
XRTS - Read task status
XSTP - Set/read task priority
XLKT - Lock task
XULT - Unlock task
XSWP - Swap to next task
XCTB - Create task block
XKTB - Kill task
XSTM - Send task message
XGTM - Get task message
XKTM - Kill task message
XGMP - Get message pointer
XSMP - Send message pointer

XSEV - Set event flag
XSEF - Set event flag w/swap
XTEF - Test event flag
XDEV - Delay set/reset event
XSUI - Suspend until interrupt

(4.2 PDOS ASSEMBLY LANGUAGE CALLS continued)

XDTV - Define trap vectors
XSUP - Enter supervisor mode
XUSP - Return to user mode
XRSR - Read status register
XLSR - Load status register
XRTE - Return from interrupt
X881 - 68881 enable

XDMP - Dump memory from stack
XRDM - Dump registers
XBUG - Debug call
XEXC - Execute PDOS call D7.W

XLER - Load error register
XERR - Return error D0 to monitor
XEXT - Exit to monitor
XEXZ - Exit to monitor with command

Level 2 consists of system support calls. Data conversion and data/time processing are their main functions. They are as follows:

System Support calls

XCBD - Convert binary to decimal
XCBH - Convert binary to hex
XCBM - Convert to decimal w/message
XCDB - Convert decimal to binary
XCBX - Convert to decimal in buffer
XCHX - Convert binary to hex in buffer

XRDT - Read date
XRTM - Read time
XRTP - Read time parameters
XFTD - Fix time & date
XPAD - Pack ASCII date
XUAD - Unpack ASCII Date
XUDT - Unpack date
XUTM - Unpack time
XWDT - Write date
XWTM - Write time

XGNP - Get next parameter

Level 3 primitives deal with console I/O. Included are commands for setting the baud rate and other characteristics of an I/O port, reading and writing characters or lines, clearing the screen, positioning the cursor, and monitoring port status.

Console I/O Calls

XGCB - Conditional get character
XGCC - Get character conditional

(4.2 PDOS ASSEMBLY LANGUAGE CALLS continued)

XGCR - Get character
XGCP - Get port character
XGLB - Get line in buffer
XGLM - Get line in monitor buffer
XGLU - Get line in user buffer
XPCB - Push command to buffer

XPBC - Put buffer to console
XPCC - Put character(s) to console
XPCL - Put CRLF
XPCR - Put character raw
XPSP - Put space to console

XPLC - Put line to console
XPDC - Put data to console
XPEL - Put encoded line to console
XPMC - Put message to console
XPEM - Put encoded message to console

XCLS - Clear screen
XPSC - Position cursor
XTAB - Tab to column
XRCP - Read port cursor position

XBCP - Baud console port
XSPF - Set port flag
XRPS - Read port status
XCBC - Check for break character
XCBP - Check for break or pause

Level 4 primitives are file support calls for the file manager. However, important functions such as copying files, appending files, sizing disks, and resetting disks are included here.

File Support Calls

XFFN - Fix file name
XLFN - Look for name in file slots
XLST - List file directory
XBFL - Build file directory list
XRDE - Read next directory entry
XRDN - Read directory entry by name

XAPF - Append file
XCPY - Copy file

XCHF - Chain command
XLDF - Load file
XRCN - Reset console inputs
XRST - Reset disk
XSZF - Get disk size

(4.2 PDOS ASSEMBLY LANGUAGE CALLS continued)

Level 5 primitives are the file management calls of PDOS. They use the file lock (event 120) to prevent conflicts between multiple tasks. Functions such as defining, deleting, reading, writing, positioning, and locking are supported by the file manager.

File Management Calls

XDFL - Define file
XRNF - Rename file
XRFA - Read file attributes
XWFA - Write file attributes
XWFP - Write file parameters
XDLF - Delete file
XZFL - Zero file
XSOP - Open sequential
XROO - Open random read only
XROP - Open random

XNOP - Open non-exclusive random
XLKF - Lock file
XULF - Unlock file

XRFP - Read file position
XRWF - Rewind file
XPSF - Position file

XRBF - Read bytes from file
XRLF - Read line from file

XWBF - Write bytes to file
XWLF - Write line to file

XFBF - Flush buffers
XFAC - File altered check

XCFA - Close file w/attribute
XCLF - Close file

The final level of primitives is for disk access via the read/write logical sector routines in the PDOS BIOS. A disk lock (event 121) is used to make these calls autonomous and prevent multiple commands from being sent to the disk controller.

Disk Access Calls

XISE - Initialize sector
XRSE - Read sector
XWSE - Write sector
XRSZ - Read sector zero

4.3.1 X881 - SAVE 68881 ENABLE

Mnemonic:	X881
Value:	$A006
Module:	MPDOSK1
Format:	X881

```
START   X881
        FMOVE.L #100,FP0
        FDIV.W  #3,FP0
```

The SAVE 68881 ENABLE sets the BIOS save flag (SVF$(A6)) thus signaling the PDOS BIOS to save and restore 68881 registers and status during context switches. The save flag is again cleared by exiting to the PDOS monitor.

See also:

Chapter 8 BIOS

Possible Errors: None

4.3.2 XAPF - APPEND FILE

```
Mnemonic:      XAPF
   Value:      $A0AA
  Module:      MPDOSF
  Format:      XAPF
                 <status error return>

Registers: In  (A1) = Source file name
               (A2) = Destination file name
```

Note: A [CTRL-C] will terminate this primitive and return error -1 in data register D0.

The APPEND FILE primitive is used to append two files together. The source and destination file names are pointed to by address registers A1 and A2, respectively. The source file is appended to the end of the destination file. The source file is not altered.

Possible Errors:

```
-1 = Break
50 = Invalid file name
53 = File not defined
60 = File space full
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors
```

```
APFL    LEA.L   SF1(PC),A1 ;SOURCE FILE NAME
        LEA.L   SF2(PC),A2 ;DESTINATION FILE NAME
        XAPF               ;APPEND
          BNE.S ERROR      ;ERROR
        ....               ;SUCCESS

SF1     DC.B    'FILE1',0
DF2     DC.B    'FILE2',0
        EVEN
```

4.3.3 XBCP - BAUD CONSOLE PORT

```
Mnemonic:     XBCP
   Value:     $A070
  Module:     MPDOSK2
  Format:     XBCP
                <status error return>

Registers: In  D2.W = fOPI 8DBS / <port #>
               D3.W = Baud rate
               D4.W = Port type
               D5.L = Port base
```

The BAUD CONSOLE PORT primitive initializes any one of the PDOS I/O ports and binds a physical UART to a character buffer. The primitive sets handshaking protocol, receiver and transmitter baud rates, and enables receiver interrupts.

Data register D2 selects the port number and sets (or clears) the corresponding flag bits. If D2.W is negative, then the absolute value is subsequently used and the port number is stored in U2P$(A6).

The right byte of data register D2 (bits 0-7) selects the console port. The left byte of D2.W (bits 8-15) selects various flag options including ^S-^Q and/or DTR handshaking, receiver parity and interrupt enable, and 8-bit character I/O.

The receiver and transmitter baud rates are initialized to the same value according to register D3. Register D3 ranges from 0 to 7 or the corresponding baud rates of 19200, 9600, 4800, 2400, 1200, 600, 300, or 110.

If data register D4 is non-zero, then it selects the port type and register D5 selects the port base address. These parameters are system-defined and correspond to the UART module. If register D4 is zero, there is no change.

See also:

4.3.84 XRPS - READ PORT STATUS
4.3.98 XSPF - SET PORT FLAG

Possible Errors:

66 = Invalid port or baud rate

```
START   MOVE.W  #$103,D2   ;PORT 3 W/^S^Q
        MOVE.W  #19200,D3  ;19.2K BAUD
        MOVEQ.L #0,D4      ;NO TYPE CHANGE
        XBCP               ;BAUD PORT
          BNE.S ERROR
        ....
```

```
F8BT. = fOPI 8DBS
         \\\\ \\\\_ 0 = ^S^Q enable
          \\\\ \\\_ 1 = Control char disable
           \\\\ \\_ 2 = DTR enable
            \\\\ \_ 3 = 8-bit character enable
             \\\\__ 4 = Receiver interrupt enable
              \\\__ 5 = Even parity enable
               \\__ 6 = *Reserved (High/low water)
                \__ 7 = **Reserved (^S^Q flag bit)

         *Used to clear all bits
        **Used to set U2P(A6)$
```

```
D3.W = Baud = 0 = 19200 baud
              1 = 9600 baud
              2 = 4800 baud
              3 = 2400 baud
              4 = 1200 baud
              5 = 600 baud
              6 = 300 baud
              7 = 110 baud
```

4.3.4 XBFL - BUILD FILE DIRECTORY LIST

```
     Mnemonic:   XBFL
        Value:   $A0B8
       Module:   MPDOSM
       Format:   XBFL
                   <status error return>

Registers: In    (A1) = List specifications
                 (A2) = Beginning buffer address
                 (A3) = End buffer address
           Out   (A3) = Updated buffer end address
```

The BUILD FILE DIRECTORY LIST primitive builds a serial list of file names in memory as selected by the list specifications. Address register A1 points to the file list specifications.

List specifications:

```
 <file list> = {file}{:ext}{;level}{/disk}{/select...}

 where   {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
         {:ext} = 1 to 3 characters (:@=all,*=wild)
       {;level} = directory level (;@=all)
        {/disk} = disk number ranging from 0 to 255
      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                  PDOS attribute (/*,/**)
                  Change date (/Fdy-mon-yr,/Tdy-mon-yr)
                        or    (/Fmn/dy/yr,/Tmn/dy/yr)
```

Address registers A2 and A3 point to the beginning and end of the memory buffer respectively. Register A3 is updated to a word boundary just after the last file name null.

Possible Errors:

Disk errors
67 = Invalid Parameter
73 = Not Enough Memory

```
GETL    LEA.L   SPC(PC),A1   ;POINT TO LIST
        LEA.L   BUF(PC),A2   ;GET BUFFER ADDRESS
        LEA.L   EBUF(PC),A3  ;GET END POINTER
        XBFL                 ;BUILD LIST
          BNE.S ERROR
*
PRNT    TST.B   (A1)         ;ENTRY?
          BEQ.S DONE         ;N
        XPCL                 ;Y, OUTPUT CRLF
        XPLC                 ;OUTPUT ENTRY
*
NEXT    TST.B   (A1)+        ;NEXT, DONE?
          BNE.S NEXT         ;N
        BRA.S   PRNT         ;Y
*
DONE    ....
*
ERROR   ....

SPC     DC.B    '@:SR;@/0',0
BUF     DS.B    500
EBUF    EQU     *
```

4.3.5 XBUG - DEBUG CALL

Mnemonic:	XBUG
Value:	$A038
Module:	MPDOSD
Format:	XBUG

Registers: None

The DEBUG CALL primitive breaks from the user program and enters the PDOS debugger. All registers are saved and you are prompted for additional commands.

The following are legal debugger commands:

A0-7	A-reg	#	Mem IAC
B{#,a}	Lst/def break	#,#	Mem dump
D0-7	D-reg	#,#+	Disassemble
F	68881 regs	#,#,#{WL}	Find B/W/L
{#}G	Go & break	#(0-7	d(Ax)
M	Last dump	#{+-}#	Hex +/-
N#	0=W,1=B,+2=w/o read		
O	Offset	^D	Disassemble
P	PC	-	Open previous
Q	Exit	LF	Open next
R	Reg dump	+#	# + offset
S	Status		
T	Trace	Trace options:	
U	Unit	--------------	
V	Control IAC	F/R/M	Dump
W{s,e}	Window	G	Go
X	Set breaks & exit	T	Running
Z	Reset		

See also:

4.3.23 XDMP - DUMP MEMORY FROM STACK
4.3.75 XRDM - DUMP REGISTERS
PB - PDOS DEBUGGER (chapter 3)

Possible Errors: None

4.3.6 XCBC - CHECK FOR BREAK CHARACTER

```
  Mnemonic:    XCBC
     Value:    $A072
    Module:    MPDOSK2
    Format:    XCBC
                 <status return>

Registers: Out SR = EQ....No break
                    LO....[CTRL-C], Clear flag & buffer
                    LT....[ESC], Clear flag
                    MI....[CTRL-C] or [ESC]

Note: If the ignore control character bit ($02)
      of the port flag is set, then XCBC always
      returns .EQ. status.
```

```
        ....
        XCBC                  ;BREAK?
          BLO.S CONTC         ;Y, ^C
          BLT.S ESCAP         ;Y, ESC
        BRA.S   LOOP          ;N, CONTINUE
*
CONTC   ....                  ;CONTROL C

        BRA.S   BEGIN         ;START AGAIN
*
ESCAP   XPMC    BRKM          ;OUTPUT '>>BREAK'
        XEXT                  ;EXIT TO PDOS
*
BRKM    DC.B    $0A,$0D       ;BREAK MESSAGE
        DC.B    '>>BREAK',0
```

The CHECK FOR BREAK CHARACTER primitive checks the current user input port break flag (BRKF.(A5)) to see if a break character has been entered. The PDOS break characters are [CTRL-C] and the [ESC] key.

A [CTRL-C] sets the port break flag to one, while an [ESC] character sets the flag to a minus one. The XCBC primitive samples and clears this flag. The condition of the break flag is returned in the status register.

An 'LO' condition indicates a [CTRL-C] has been entered. The break flag and the input buffer are cleared. All subsequent characters entered after the [CTRL-C] and before the XCBC call are dropped. All open procedure files are closed and any system frames are restored. Also, the last error number flag (LEN$) is set to -1 and a '^C' is output to the port.

An 'LT' condition indicates an [ESC] character has been entered. Only the break flag is cleared and not the input buffer. Thus, the [ESC] character remains in the buffer.

The [CTRL-C] character is interpreted as a hard break and is used to terminate command operations. The [ESC] character is a soft break and remains in the input buffer, even though the break flag is cleared by the XCBC primitive. (This allows an editor to use the [ESC] key for special functions or command termination.)

Note: If the ignore control character bit ($02) of the port flag is set, then XCBC always returns .EQ. status.

Possible Errors: None

4.3.7 XCBD - CONVERT BINARY TO DECIMAL

```
    Mnemonic:    XCBD
       Value:    $A050
      Module:    MPDOSK3
      Format:    XCBD

  Registers: In  D1.L = Number
            Out  (A1) = String
```

The CONVERT BINARY TO DECIMAL primitive converts a 32-bit, 2's complement number to a character string. The number to be converted is passed to XCBD in data register D1. Address register A1 is returned with a pointer to the converted character string located in the monitor work buffer (MWB$).

Leading zeros are suppressed and a negative sign is the first character for negative numbers. The string is delimited by a null. The string has a maximum length of 11 characters and ranges from -2147483648 to 2147483647.

See also:

4.3.11 XCBX - CONVERT TO DECIMAL IN BUFFER.

Possible Errors: None

```
         MOVE.L  #1234,D1 ;GET NUMBER
         XCBD             ;CONVERT TO PRINT
         XPLC             ;PRINT
         ....
```

```
*******************************************
* OUTPUT LEFT JUSTIFIED NUMBER
*
*        D0.W = # OF PLACES
*        D1.L = NUMBER
*
LEFT     MOVEM.L D0/A0-A1,-(A7)
         XCBD            ;CONVERT
         MOVEA.L A1,A0   ;GET POINTER
*
LEFT02   SUBQ.W  #1,D0   ;COUNT LENGTH
         TST.B   (A0)+   ;END?
           BNE.S LEFT02  ;N
*
LEFT04   XPSP            ;OUTPUT SPACE
         SUBQ.W  #1,D0   ;DONE?
           BPL.S LEFT04  ;N
         XPLC            ;Y, OUTPUT #
         MOVEM.L (A7)+,D0/A0-A1
         RTS
```

4.3.8 XCBH - CONVERT BINARY TO HEX

```
        Mnemonic:    XCBH
           Value:    $A052
          Module:    MPDOSK3
          Format:    XCBH

      Registers: In  D1.L = Number
                Out  (A1) = String
```

The CONVERT BINARY TO HEX primitive converts a 32-bit number to its hexadecimal (base 16) representation. The number is passed in data register D1 and a pointer to the ASCII string is returned in address register A1. The converted string is found in the monitor work buffer (MWB$) of the task control block and consists of eight hexadecimal characters followed by a null.

See also:

4.3.15 XCHX - CONVERT BINARY TO HEX IN BUFFER.

Possible Errors: None

```
          MOVEQ.L #123,D1 ;GET NUMBER
          XCBH            ;GET HEX CONVERSION
          MOVEQ.L #'$',D0 ;ADD HEX SIGN
          XPCC            ;PRINT
          XPLC            ;PRINT 8 HEX CHARACTERS
          ....

***********************************************
*         DUMP REGISTERS ON USER STACK
*
*         USP = A7 = RETURN PC
*                    D0-D7
*                    A0-A7
*
DMRG      MOVEA.L (A7)+,A0 ;GET RETURN ADR
          MOVE.L  #$0007BCF7,D4
          MOVE.W  #'0D',D0
*
DMRG02    XPCL             ;OUT CRLF
          XPCC             ;OUT LINE TYPE
          MOVE.W  #' :',D0
*
DMRG04    XPCC             ;OUT DELIMITER
          MOVE.L  (A7)+,D1 ;GET REGISTER
          XCBH             ;CONVERT
          XPLC             ;OUTPUT
          MOVEQ.L #' ',D0  ;CHANGE TO ' '
          LSR.L   #1,D4    ;4 DONE?
            BCS.S DMRG04   ;N
          XPCC             ;Y, OUT SPACE
          LSR.L   #1,D4    ;CRLF?
            BCS.S DMRG04   ;N
          MOVE.W  #'0A',D0 ;Y, CHANGE TO 'A'
          LSR.L   #1,D4    ;MORE?
            BCS.S DMRG02   ;Y
          JMP     (A0)     ;N, RETURN
```

4.3.9 XCBM - CONVERT TO DECIMAL W/MESSAGE

```
      Mnemonic:   XCBM
         Value:   $A054
        Module:   MPDOSK3
        Format:   XCBM     <message>

     Registers: In  D1.L = Number
               Out  (A1) = String
```

The CONVERT TO DECIMAL WITH MESSAGE primitive converts a 32-bit, signed number to a character string. The output string is preceded by the string whose PC relative address is in the operand field of the call.

The string can be up to 20 characters in length and is terminated by a null character. The number to be converted is passed to XCBM in data register D1. Address register A1 is returned with a pointer to the converted character string which is located in the monitor work buffer (MWB$) of the task control block.

Leading zeros are suppressed and the result ranges from -2147483648 to 2147483647.

The message address is a signed 16-bit PC relative address.

Possible Errors: None

```
START   MOVE.L  #$80000004,D1
*
LOOP    XPMC    MES1      ;HEADING
        XCBH              ;CONVERT HEX
        XPLC
        XCBM    MES2      ;CONVERT DECIMAL
        XPLC
        SUBQ.L  #1,D1
        CMPI.L  #$7FFFFFFC,D1
        BHS.S   LOOP
        XEXT
*
MES1    DC.B    $0A,$0D,'Hex $',0
MES2    DC.B    ' = ',0
        EVEN
        END     START

x>TEST
Hex $80000004 = -2147483644
Hex $80000003 = -2147483645
Hex $80000002 = -2147483646
Hex $80000001 = -2147483647
Hex $80000000 = -2147483648
Hex $7FFFFFFF = 2147483647
Hex $7FFFFFFE = 2147483646
Hex $7FFFFFFD = 2147483645
Hex $7FFFFFFC = 2147483644
x>
```

4.3.10 XCBP - CHECK FOR BREAK OR PAUSE

```
Mnemonic:      XCBP
   Value:      $A074
  Module:      MPDOSK2
  Format:      XCBP
                 <status return>

Registers: Out SR = EQ...No character
                    LT...[ESC]
                    LO...[CTRL-C]
                    NE...Pause
```

Note: If a 'BLT' instruction does not immediately follow the XCBP call, then the primitive exits to PDOS when an [ESC] character is entered.

If the ignore control character bit ($02) of the port flag is set, then XCBP always returns .EQ. status.

```
LOOP    ....             ;OUTPUT

        XCBP             ;LOOK FOR PAUSE
          BLT.S EXIT     ;ESC
        BRA.S   LOOP     ;CONTINUE
*
EXIT    ....             ;ESC
```

The CHECK FOR BREAK OR PAUSE primitive looks for a character from your PRT$(A6) port. Any non-control character will cause XCBP to output a pause message and wait for another character.

The pause message consists of:

```
[CR]
'Strike any key...'
[CR]
'
[CR].
```

A [CTRL-C] will abort any assigned console file and return the status 'LO'. If a 'BLT' instruction follows the XCBP primitive and an [ESC] character is entered, then the call returns with status 'LT'. Otherwise, an [ESC] will abort your program to the PDOS monitor.

An 'EQ' status indicates that no character was entered. An 'NE' status indicates a pause has occurred.

Possible Errors: None

4.3.11 XCBX - CONVERT TO DECIMAL IN BUFFER

Mnemonic:	XCBX
Value:	$A06A
Module:	MPDOSK3
Format:	XCBX

Registers: In D1.L = Number
(A1) = Buffer

The CONVERT TO DECIMAL IN BUFFER primitive converts a 32-bit, 2's complement number to a character string. The number to be converted is passed to XCBX in data register D1. Address register A1 points to the buffer where the converted string is stored.

Leading zeros are suppressed and a negative sign is the first character for negative numbers. The string is delimited by a null. The string has a maximum length of 11 characters and ranges from -2147483648 to 2147483647.

See also:

4.3.7 XCBD - CONVERT BINARY TO DECIMAL.

Possible Errors: None

```
        MOVEA.L A6,A1   ;POINT TO USER BUF
        MOVEQ.L #12,D1  ;GET #
        BSR.S   OUTS    ;OUTPUT TO BUFFER
        XPBC            ;OUTPUT BUFFER
        ....

OUTS    XCBX            ;CONVERT #
*
OUTS02  TST.B   (A1)+   ;END?
         BNE.S OUTS02   ;N
        SUBQ.W  #1,A1   ;Y, BACKUP
        RTS             ;RETURN
```

4.3.12 XCDB - CONVERT ASCII TO BINARY

```
Mnemonic:       XCDB
   Value:       $A056
  Module:       MPDOSK3
  Format:       XCDB
                  <status return>

Registers: In   (A1) = String
          Out   D0.B = Delimiter
                D1.L = Number
                (A1) = Updated string
                  SR = LT....No number
                       EQ....# w/o null delimiter
                       GT....#
```

Note: XCDB does not check for overflow.

The CONVERT ASCII TO BINARY primitive converts an ASCII string of characters to a 32-bit, 2's complement number. The result is returned in data register D1 while the status register reflects the conversion results.

XCDB converts signed decimal, hexadecimal, or binary numbers. Hexadecimal numbers are preceded by "$" and binary numbers by "%". A "-" indicates a negative number. There can be no embedded blanks.

An 'LT' status indicates that no conversion was possible. Data register D0 is returned with the first character and address register A1 points immediately after it.

A 'GT' status indicates that a conversion was made with a null delimiter encountered. The result is returned in data register D1. Address register A1 is returned with an updated pointer and register D0 is set to zero.

An 'EQ' status indicates that a conversion was made but the ASCII string was not terminated with a null character. The result is returned in register D1 and the non-numeric, non-null character is returned in register D0. Address register A2 has the address of the next character.

Possible Errors: None

```
START   MOVEQ.L #0,D5   ;GET DEFAULT
        XPMC    MES1    ;OUTPUT PROMPT
        XGLU            ;GET REPLY
          BLS.S STRT04  ;USE DEFAULT
        XCDB            ;CONVERT, OK?
          BGT.S STRT02  ;Y
        XPMC    ERM1    ;N, REPORT
        BRA.S   START   ;TRY AGAIN
*
STRT02  MOVE.L  D1,D5   ;SAVE VALUE
        ....
STRT04

MES1    DC.B    $0A,$0D,'ANSWER=',0
ERM1    DC.B    $0A,$0D,'INVALID!',0
        EVEN
```

4.3.13 XCFA - CLOSE FILE W/ATTRIBUTE

```
     Mnemonic:     XCFA
        Value:     $A0D0
       Module:     MPDOSF
       Format:     XCFA
                     <status error return>

Registers: In      D1.W = File ID
                   D2.B = New attribute
```

```
MOVE.W  D5,D1    ;GET FILE ID
MOVE.B  #$20,D2  ;CLOSE AS OBJECT
XCFA             ;CLOSE FILE
  BNE.S ERROR
....
```

The CLOSE FILE WITH ATTRIBUTES primitive closes the open file specified by data register D1. At the same time, the file attributes are updated according to the byte contents of data register D2.

If the file was opened for sequential access and the file has been updated, then the END-OF-FILE marker is set at the current file pointer. If the file was opened for random or shared access, then the END-OF-FILE marker is updated only if the file has been extended (data was written after the current END-OF-FILE marker).

The LAST UPDATE is updated to the current date and time only if the file has been altered.

All files must be closed when opened! Otherwise, directory information and possibly even the file itself will be lost.

*Note: If the file is not altered, then XCFA will not alter the file attributes.

```
D2.B = $80   AC or Procedure file
     = $40   BN or Binary file
     = $20   OB or 68000 object file
     = $10   SY or 68000 memory image
     = $08   BX or BASIC binary token file
     = $04   EX or BASIC ASCII file
     = $02   TX or Text file
     = $01   DR or System I/O driver
     = $00   Clear file attributes

D1.W = File ID = (Disk #) x 256 + (File slot index)
```

See also:

4.3.79 XRFA - READ FILE ATTRIBUTES
4.3.115 XWFA - WRITE FILE ATTRIBUTES
4.3.116 XWFP - WRITE FILE PARAMETERS

Possible Errors:

52 = File not open
59 = Invalid file slot
75 = File locked
Disk errors

4.3.14 XCHF - CHAIN COMMAND

Mnemonic: XCHF
Value: $AOAC
Module: MPDOSM
Format: XCHF

Registers: In A1.L = File name

Note: The primitive returns only on error.

```
        LEA.L   FILEN(PC),A1 ;GET FILE NAME
        XCHF                 ;CHAIN FILE
        XERR                 ;PROBLEM
*
FILEN   DC.B    'NEXTPRGM',0
        EVEN
```

The CHAIN FILE primitive is used by the PDOS monitor to execute program files. The primitive chains from one program to another according to the file type.

Address register A1 points to the chain file name. The file type determines how the file is to be executed. If the file is typed 'OB' or 'SY', then the 68000 loader is called (XLDF). If the file is typed 'BX' or 'EX', then the PDOS BASIC interpreter loads the file and begins executing at the lowest line number. Likewise, if the file is typed 'AC', then control returns back to the PDOS monitor and further requests for console characters reference the file.

The XCHF call returns only if an error occurs during the chain operation. All other errors, such as those occurring in BASIC, return to the PDOS monitor.

Parameters may be passed from one program to another through the user TEMP variables located in the task control block or through the system messages buffers.

See also:

4.3.28 XEXZ - EXIT TO MONITOR W/COMMAND

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
62 = No start address
63 = Illegal object tag
64 = Illegal section
65 = File not loadable
71 = Nesting error
77 = Procedure not memory resident
Disk errors

4.3.15 XCHX - CONVERT BINARY TO HEX IN BUFFER

Mnemonic:	XCHX
Value:	$A068
Module:	MPDOSK3
Format:	XCHX
Registers: In	D1.L = Number
	(A1) = Output buffer

The CONVERT BINARY TO HEX IN BUFFER primitive converts a 32-bit number to its hexadecimal (base 16) representation. The number is passed in data register D1 and a pointer to a buffer in address register A1. The converted string consists of eight hexadecimal characters followed by a null.

See also:

4.3.8 XCBH - CONVERT BINARY TO HEX.

Possible Errors: None

```
START   MOVE.L  #$80000004,D1
*
LOOP    MOVEA.L A6,A1     ;USER BUFFER
        BSR.S   OUTS      ;OUT HEADING
          DC.W  MES1-*
        XCHX              ;CONVERT HEX
*
LOOP2   TST.B   (A1)+     ;END?
          BNE.S LOOP2     ;N
        SUBQ.W  #1,A1     ;Y
        BSR.S   OUTS      ;' = '
          DC.W  MES2-*
        XCBX              ;CONVERT DECIMAL
*
LOOP4   TST.B   (A1)+     ;END?
          BNE.S LOOP4     ;N
        XPBC              ;Y, OUTPUT
        SUBQ.L  #1,D1
        CMPI.L  #$7FFFFFFC,D1
          BHS.S LOOP
        XEXT
*
OUTS    MOVEA.L (A7),A0 ;GET ADDRESS
        ADDQ.L  #2,(A7) ;ADJUST PC
        ADDA.W  (A0)+,A0
*
OUTS2   MOVE.B  (A0)+,(A1)+
          BNE.S OUTS2
        SUBQ.W  #1,A1
        RTS
*
MES1    DC.B    $0A,$0D,'Hex $',0
MES2    DC.B    ' = ',0
        EVEN
        END     START

x>TEST
Hex $80000004 = -2147483644
Hex $80000003 = -2147483645
Hex $80000002 = -2147483646
Hex $80000001 = -2147483647
Hex $80000000 = -2147483648
Hex $7FFFFFFF = 2147483647
Hex $7FFFFFFE = 2147483646
Hex $7FFFFFFD = 2147483645
Hex $7FFFFFFC = 2147483644
x>
```

4.3.16 XCLF - CLOSE FILE

```
Mnemonic:     XCLF
   Value:     $A0D2
  Module:     MPDOSF
  Format:     XCLF
                <status error return>

Registers: In   D1.W = File ID
```

The CLOSE FILE primitive closes the open file as specified by the file ID in data register D1. If the file was opened for sequential access and the file was updated, then the END-OF-FILE marker is set at the current file pointer.

If the file was opened for random or shared access, then the END-OF-FILE marker is updated only if the file was extended (ie. data was written after the current END-OF-FILE marker).

If the file has been altered, the current date and time is stored in the LAST UPDATE variable of the file directory.

All open files must be closed at or before the completion of a task (or before disks are removed from the system)! Otherwise, directory information is lost and possibly even the file itself.

Possible Errors:

```
52 = File not open
59 = Invalid slot #
75 = File locked
Disk errors
```

```
        MOVE.W  D5,D1      ;GET FILE ID
        XCLF               ;CLOSE FILE
          BNE.S ERROR
        ....

ERROR   CLR.L   D1
        MOVE.W  D0,D1      ;GET ERROR #
        XCBM    ERM1       ;CONVERT
        XPLC               ;OUTPUT
        ....

ERM1    DC.B    $0A,$0D
        DC.B    'PDOS CLOSE ERR ',0
        EVEN
```

File ID = (Disk #) x 256 + (File slot index)

4.3.17 XCLS - CLEAR SCREEN

```
Mnemonic:   XCLS
   Value:   $A076
  Module:   MPDOSK2
  Format:   XCLS

Registers:  None
```

```
....
XCLS            ;CLEAR SCREEN
XPMC    MES01   ;OUTPUT MESSAGE
....
```

Note: The clear screen characters are located in the user TCB variable CSC$(A6).

The CLEAR SCREEN primitive clears the console screen, homes the cursor, and clears the column counter. This function is adapted to the type of console terminals used in the PDOS system.

The character sequence to clear the screen is located in the task control block variable CSC$(A6). These characters are transferred from the parent task to the spawned task during creation. The initial characters come from the BIOS module.

If CSC$ is nonzero, then the CLEAR SCREEN primitive outputs up to four characters: one or two characters; an [ESC] followed by a character; or an [ESC], character, [ESC], and a final character. The one-word format allows for two characters. The parity bits cause the [ESC] character to precede each character.

```
CSC$(A6) = E111 1111 E222 2222
           \\     \ \\     \_
            \\     \ \\_______ 2nd character
             \\     \ \_______ 2nd [ESC]
              \\     \
               \\     \_______
                \\____________ 1st character
                 \____________ 1st [ESC]
```

If CSC$ is zero, then PDOS makes a call into the BIOS for custom clear screens. The entry point is B_CLS beyond the BIOS table.

The MTERM utility normally maintains the CSC$ field, although it can be altered under program control. The initial definition of CSC$ is found in the MBIOS:SR file and can be modified by doing a new SYSGEN.

```
MOVE.W  CSC$(A6),D0     ;GET CLEAR CHARACTERS, ESC?
  BLT.S @0002           ;Y, INSERT ESC
  BGT.S @0004           ;N
MOVEA.L (A5),A0         ;N, USE BIOS
JSR     B_CLS(A0)       ;CLEAR SCREEN
  BNE.S @0008
```

See also:

4.3.73 XRCP - READ PORT CURSOR POSITION
CHAPTER 8 - BIOS

Possible Errors: None

4.3.18 XCPY - COPY FILE

Mnemonic: XCPY
Value: $A0AE
Module: MPDOSF
Format: XCPY
<status error return>

Registers: In (A1) = Source file name
(A2) = Destination file name

Note: A [CTRL-C] terminates this primitive and returns the error -1 in register D0.

```
        LEA.L   FILES(PC),A1 ;SOURCE FILE NAME
        LEA.L   FILED(PC),A2 ;DEST. FILE NAME
        XCPY                 ;COPY FILE
          BNE.S ERROR        ;PROBLEM
        ....                 ;CONTINUE

FILES   DC.B    'TEMP',0
FILED   DC.B    'TEMP:BK/1',0
        EVEN
```

The COPY FILE primitive copies the source file into the destination file. The source file is pointed to by address register A1 and the destination file is pointed to by register A2. A [CTRL-C] halts the copy, prints '^C' to the console, and returns with error -1.

The file attributes of the source file are automatically transferred to the destination file.

Possible Errors:

-1 = Break file transfer
50 = Invalid file name
53 = File not defined
60 = File space full
61 = File already open
68 = Not PDOS disk
69 = No more file slots
70 = Position error
Disk errors

4.3.19 XCTB - CREATE TASK BLOCK

Mnemonic: XCTB
Value: $A026
Module: MPDOSK1
Format: XCTB
<status error return>

Registers: In D0.W = Task size (1k byte increments)
D1.W = Task time.B/priority.B
D2.W = I/O port
(A0) = Optional low memory pointer
(A1) = Optional high memory pointer
(A2) = Command line pointer or entry address
Out D0.L = Spawned task number

Note: If D0.W is positive, A0 and A1 are undefined.

If D0.W equals zero, then A0 and A1 are the new task's memory bounds and A2 contains the task's entry address.

If D0.W is negative, then A0 and A1 are the new task's memory bounds and A2 points to the task's command line.

The CREATE TASK primitive places a new task entry in the PDOS task list. Memory for the new task comes from either the parent task or the system memory bit map. Data register D0 controls the creation mode of the new task as well as the task size.

If register D0.W is positive, then the first available contiguous memory block equal to D0.W (in 1K bytes) is allocated to the new task. If there is not a block big enough, then the upper memory of the parent task is allocated to the new task. The parent task's memory is then reduced by D0.W x 1K bytes. Address register A2 points to the new task command line. If A2 is zero, then the monitor is invoked.

If register D0.W is zero, then registers A0 and A1 specify the new task's memory limits. Register A2 specifies the task's starting PC. The task control block begins at (A0) and is immediately followed by an XEXT primitive. The task user stack pointer is set at (A1). Thus, the new program should allow $502 bytes at the low end and enough user stack space at the upper end.

Continued on next page...

If D0>0 then: D0=Task size
(A2)=Task command line
(0=Monitor)

```
MOVEQ.L #10,D0   ;10 K BYTES
MOVEQ.L #64,D1   ;PRIORITY 64
MOVEQ.L #1,D2    ;PORT 1
SUBA.L  A2,A2    ;CALL MONITOR
XCTB             ;CREATE TASK
  BNE.S ERROR
```

If D0=0 then: (A2)=Task entry address
A0-A1=New task memory limits

```
MOVEQ.L #0,D0       ;USE A0-A1 BOUNDS
MOVEQ.L #64,D1      ;PRIORITY 64
MOVEQ.L #1,D2       ;PORT 1
LEA.L   SRAM,A0     ;TCB ADDR (START)
LEA.L   ERAM,A1
LEA.L   P(PC),A2 ;PC
XCTB                ;CREATE TASK
```

(4.3.19 XCTB - CREATE TASK BLOCK continued)

If data register D0.W is negative, then registers A0 and A1 specify the new task's memory limits. Register A2 points to the new task command line. (If A2=0, then the monitor is invoked.)

If D0=<0 then: (A2)=Task command line (0=Monitor)
A0-A1=New task memory limits

```
        MOVEQ.L #0,D0     ;USE A0-A1 BOUNDS
        MOVEQ.L #64,D1    ;PRIORITY 64
        MOVEQ.L #1,D2     ;PORT 1
        LEA.L   SRAM,A0   ;TCB ADDR (START)
        LEA.L   ERAM,A1
        LEA.L   C(PC),A2  ;PC
        XCTB              ;CREATE TASK
          BNE.S ERROR

C       DC.B    'PRGM1',0
```

The command line is transferred to the spawned program via a system message buffer. The maximum length of a command line is 64 characters. When the task is scheduled for the first time, the message buffers are searched for a command. Messages with a source task equal to $FF are considered commands and moved to the task's monitor buffer. The task CLI then processes the line. If no command message is found, then the monitor is called directly.

Data register D1.W specifies the new task's priority. The range is from 1 to 255. The larger the number, the higher the priority.

D1=Task priority

Data register D2.W specifies the I/O port to be used by the new task. If register D2.W is positive, then the port is available for both input and output. If register D2.W is negative, then the port is used only for output. If register D2.W is zero, then no port is assigned. Only one task may be assigned to any one input port while many tasks may be assigned to an output port. Hence, a port is allocated for input only if it is available. An invalid port assignment does not result in an error.

D2=I/O port

If D2=0, then phantom port (no I/O)

If D2>0, then port is used for I/O

If D2<0, then port is used for output only

A call is made to D$INT in the debugger module. This initializes all addresses, registers, breaks, and offsets.

Finally, the spawned task's number is returned in register D0.L to the parent task. This can be used later to test task status or to kill the task.

Possible Errors:

72 = Too many tasks
73 = Not enough memory

4.3.20 XDEV - DELAY SET/RESET EVENT

```
Mnemonic:      XDEV
   Value:      $A032
  Module:      MPDOSK1
  Format:      XDEV
                 <status error return>

Registers: In  DO.L = Time
               D1.B = Event (+=Set, -=Reset)
```

Note: If DO.L=0, then the D1.B event is cleared.

The DELAY SET/RESET EVENT primitive places a timed event in a system stack controlled by the system clock. Data register DO.L specifies the time interval in clock tics. When it counts to zero, then the event D1.B is set if positive, or reset if negative.

If the event already exists in the stack, it is replaced by the new entry. If the time specified in DO equals zero, then any pending timed event equal to D1.B is deleted from the stack.

If D1.B is positive, event D1.B is first cleared. If D1.B is negative, event D1.B is set before exiting the primitive.

See also:

4.3.94 XSEF - SET EVENT FLAG W/SWAP
4.3.95 XSEV - SET EVENT FLAG
4.3.101 XSUI - SUSPEND UNTIL INTERRUPT
4.3.106 XTEF - TEST EVENT FLAG

Possible Errors:

83 = Delay event stack full

```
GETC   XGCC                    ;CHARACTER?
         BNE.S GETC2           ;Y
       MOVEQ.L #100,DO         ;N, GET DELAY
       MOVE.L  #128,D1         ;USER LOCAL EVENT
       XDEV                    ;DELAY 128 1 SECOND
         BNE.S GETC            ;FULL
       LSL.W   #8,D1           ;GET 128/(PORT+96)
       MOVE.B  #96,D1
       ADD.B   PRT$(A6),D1
       XSUI                    ;SUSPEND
       CMP.B   DO,D1           ;CHARACTER EVENT?
         BEQ.S GETC            ;Y
       XRTM                    ;N, READ TIME
       MOVE.B  7(A1),DO        ;GET LAST CHARACTER
       CMP.B   T(A6),DO        ;SAME TIME?
         BEQ.S GETC            ;Y, TRY AGAIN
       MOVE.L  (A1)+,T(A6)     ;N, SAVE NEW TIME
       MOVE.L  (A1),T+4(A6)
       CLR.B   T+8(A6)
       BSR.S   POSIT           ;POSITION & OUTPUT TIME
         DC.W  23*256+11
         DC.W  0
       BRA.S   GETC            ;TRY AGAIN
```

4.3.21 XDFL - DEFINE FILE

```
Mnemonic:     XDFL
   Value:     $A0D4
  Module:     MPDOSF
  Format:     XDFL
                <status error return>

Registers: In D0.W = # of contiguous sectors
              (A1) = File name
```

The DEFINE FILE primitive creates a new file entry in a PDOS disk directory, specified by address register A1. A PDOS file name consists of an alphabetic character followed by up to 7 additional characters. An optional 3 character extension can be added if preceded by a colon. Likewise, the directory level and disk number are optionally specified by a semicolon and slash respectively. The file name is terminated with a null.

Data register D0 contains the number of sectors to be initially allocated at file definition. If register D0 is nonzero, then a contiguous file is created with D0 sectors. Otherwise, only one sector is allocated. Each sector of allocation corresponds to 252 bytes of data.

A contiguous file facilitates random access to file data since PDOS can directly position to any byte within the file without having to follow sector links. A contiguous file is automatically changed to a non-contiguous file if it is extended with non-contiguous sectors.

Possible Errors:

- 50 = Invalid file name
- 51 = File already defined
- 55 = Fragmentation error
- 57 = File directory full
- 61 = File already open
- 68 = Not PDOS disk
- Disk errors

```
        CLR.L   D0          ;SEQUENTIAL FILE
        LEA.L   FN(PC),A1   ;GET FILE NAME
        XDFL                ;DEFINE FILE
          BNE.S ERROR       ;ERROR
        ....

        MOVEQ.L #100,D0     ;RANDOM ACCESS FILE
        LEA.L   FN(PC),A1   ;GET FILE NAME
        XDFL                ;DEFINE CONTIGUOUS
          BNE.S ERROR
        ....

FN      DC.B    'FILENAME:EXT',0
        EVEN
```

D0.W > 0 Contiguous file with D0 sectors

D0.W = 0 Non-contiguous file

4.3.22 XDLF - DELETE FILE

```
Mnemonic:   XDLF
   Value:   $A0D6
  Module:   MPDOSF
  Format:   XDLF
              <status error return>

Registers: In   (A1) = File name
```

The DELETE FILE primitive removes the file whose name is pointed to by address register A1 from the disk directory and releases all sectors associated with that file for use by other files on that same disk. A file cannot be deleted if it is delete (*) or write (**) protected.

Possible Errors:

```
50 = Invalid file name
53 = File not defined
58 = File delete or write protected
61 = File already open
68 = Not PDOS disk
Disk errors
```

```
        LEA.L   FN(PC),A1   ;GET FILE NAME PTR
        XDLF                ;DELETE FILE
          BNE.S ERROR       ;ERROR
        ....                ;NORMAL RETURN

FN      DC.B    'TEMP/2',0
        EVEN
```

4.3.23 XDMP - DUMP MEMORY FROM STACK

```
   Mnemonic:    XDMP
      Value:    $A04A
     Module:    MPDOSK3
     Format:    XDMP

  Registers: In   USP.L = <# of bytes>.W
                          <start address>.L
            Out   USP.L = USP.L + 6
```

The DUMP MEMORY FROM STACK primitive dumps a block of memory to the console as specified by two parameters on the user stack (USP). The left side of the output is a hexadecimal dump and the right side is a masked ($7F) ASCII dump.

To use this primitive, first push a 32-bit address and then a 16-bit number of the amount of memory to be dumped. The primitive will automatically clean up the user stack.

See also:

4.3.5 XBUG - DEBUG CALL
4.3.75 XRDM - DUMP REGISTERS
PB - PDOS DEBUGGER (chapter 3)

Possible Errors: None

Example:

```
1  0/00000000:487AFFFE          START   PEA.L   START(PC)
2  0/00000004:3F3C0020                  MOVE.W  #32,-(A7)
3  0/00000008:A04A                      XDMP
4  0/0000000A:A00E                      XEXT
5  0/0000000C:        0/00000000        END     START

x>TEMP
00000D00: 487A FFFE 3F3C 0020 A04A A00E 044F 5248 Hz..?<. .J...ORH
00000D10: 20CC 20C9 43EE 068E 4298 B1C9 65FA 2D49  . .C...B...e.-I
x>
```

4.3.24 XDTV - DEFINE TRAP VECTORS

```
Mnemonic:     XDTV
   Value:     $A024
  Module:     MPDOSK1
  Format:     XDTV

Registers: In   D1.L = TVCZ FEDC BA98 7654 3210
                (A0) = Table base address
                (A1) = Vector table address

Vector table:   DC.L TRAP #0-<BASE ADR>
                ....
                DC.L TRAP #15-<BASE ADR>
                DC.L ZDIV-<BASE ADR>
                DC.L CHK-<BASE ADR>
                DC.L TRAPV-<BASE ADR>
                DC.L TRACE-<BASE ADR>
```

Note: The vector table size is variable and each entry corresponds to non-zero bits in the mask register (D1.L). Each entry is a long signed displacement from the base address register.

The DEFINE TRAP VECTORS primitive loads user routine addresses into the task control block exception vector variables. Each task has the option to process its own TRAP, zero divide, CHK, TRAPV, and/or trace exceptions.

Data register D1 selects which vectors are to be loaded according to individual bits corresponding to vectors in the vector table pointed to by address register A1. Bits 0 through 19 (right to left) correspond to TRAPs 0 through 15, zero divide, CHK, TRAPV, and trace exceptions. A 1 bit moves a vector from the vector table (biased by base address A0) into the task control block.

When an exception occurs, the task control block is checked for a corresponding non-zero exception vector. If found, then the return address is pushed on the user stack (USP) followed by the exception address and condition codes. PDOS next moves to user mode and executes a return with condition codes (RTR). This effectively acts like a jump subroutine with the return address on the user stack.

Continued on next page...

```
*                    TVCZFEDCBA9876543210
VCON    EQU         %11111000000000100001
SVECT   MOVE.L      #VCON,D1  ;GET CONTROL VAR
        LEA.L       VT(PC),A0 ;POINT TO TABLE
        MOVEA.L     A0,A1     ;BASE=TABLE
        XDTV                  ;SET VECTORS
        ....

VT      DC.L        TRAP00-VT ;TRAP #0
        DC.L        TRAP05-VT ;TRAP #5
        DC.L        TRAP15-VT ;TRAP #15
        DC.L        ZDIV-VT   ;ZERO DIVIDE
        DC.L        CHKP-VT   ;CHK PROCESSOR
        DC.L        TRPV-VT   ;TRAPV PROCESSOR
        DC.L        TRCE-VT   ;TRACE
```

```
D1.L = TVCZ FEDCBA9876543210
       \\\\ \               \_
        \\\\ \_______________ TRAPs #0-#15
         \\\\_______________ Zero divide
          \\\_______________ CHK
           \\_______________ TRAPV
            \_______________ Trace exception
```

```
IF <excp>$(A6)  THEN 1) Push return on USP
                     2) Push xxx$(A6) on USP
                     3) Push CCs on USP
                     4) Move to user mode
                     5) Exit with RTR
                ELSE PDOS error routine
```

(4.3.24 XDTV - DEFINE TRAP VECTORS continued)

The trace processing is handled differently. If the processor is in supervisor mode when a trace exception occurs, the trace bit is cleared and the exception is dismissed. The processor remains in supervisor mode. If the processor is in user mode and there is a non-zero trace variable in the task control block, then the trace is again disabled, the trace processor address is pushed on the supervisor stack along with status, and a return from exception is executed (RTE).

```
IF <sup> THEN 1) Disable trace
              2) Exit in supervisor mode
     ELSE IF TRC$(A6) THEN 1) Disable trace
                           2) Leave on stack
                           3) Push TRC$(A6)
                           4) Push SR+$2000
                           5) Exit with RTE
                       ELSE PDOS error routine
```

Possible Errors: None

4.3.25 XERR - RETURN ERROR D0 TO MONITOR

Mnemonic: XERR
Value: $A00C
Module: MPDOSK1
Format: XERR

Registers: In D0.W = Error code

The RETURN ERROR D0 TO MONITOR primitive exits to the PDOS monitor and passes an error code in data register D0. PDOS prints 'PDOS ERR', followed by the decimal error number.

The error call can be intercepted by changing the value of the ERR$ variable in the task TCB. This allows you to customize your own monitor.

See also:

4.3.27 XEXT - EXIT TO MONITOR
4.3.28 XEXZ - EXIT TO MONITOR W/COMMAND

Possible Errors: None

```
         XRSE             ;READ SECTOR
           BNE.S RERR     ;ERROR
         ....

RERR     CMPI.W  #56,D0   ;EOF?
           BNE.S RERR2    ;N
         XCLF             ;Y, CLOSE FILE
           BNE.S RERR2
         RTS
*
RERR2    XERR             ;RETURN ERROR
```

4.3.26 XEXC - EXECUTE PDOS CALL D7.W

Mnemonic: XEXC
Value: $A030
Module: MPDOSK1
Format: XEXC

Registers: In D7.W = Aline PDOS CALL

The EXECUTE PDOS CALL D7.W primitive executes a variable PDOS primitive contained in data register D7. Any registers or error conditions apply to the corresponding PDOS call.

Possible Errors: Call dependent

```
************************************
*       APPEND FILE
*
*       AF <file1>,<file2>
*
APDF    MOVE.W  #XAPF$,D7 ;APPEND COMMAND
        BRA.S   RNFLO2
*
************************************
*       COPY FILE
*
*       CF <file1>,<file2>
*
CPYF    MOVE.W  #XCPY$,D7 ;COPY COMMAND
        BRA.S   RNFLO2
*
************************************
*       RENAME FILE
*
*       RN <file1>,<file2>
*
RNFL    MOVE.W  #XRNF$,D7 ;RENAME COMMAND
*
RNFLO2  XGNP              ;SOURCE FILE
          BLE.S ERR67
        MOVEA.L A1,A2     ;SAVE
        XGNP              ;DESTINATION FILE
          BLE.S ERR67
        EXG.L   A1,A2
        XEXC              ;EXECUTE D7.W
          BNE.S RNFLO4    ;ERROR
        XEXT              ;RETURN
*
ERR67   MOVEQ.L #67,D0    ;PARAMETER ERROR
*
RNFLO4  XERR              ;ERROR
```

4.3.27 XEXT - EXIT TO MONITOR

Mnemonic:	XEXT
Value:	$A00E
Module:	MPDOSK1
Format:	XEXT
	(Always exits to monitor)
Registers:	None

```
XCLF            ;CLOSE FILE, ERROR?
  BNE.S ERROR   ;Y, DO ERROR CALL
XEXT            ;N, RETURN TO MONITOR
```

The EXIT TO MONITOR primitive exits a user program and returns to the PDOS monitor.

The exit can be intercepted by changing the value of the EXT$ variable in the task TCB. This primitive allows you to customize your own monitor.

See also:

4.3.25 XERR - RETURN ERROR DO TO MONITOR
4.3.28 XEXZ - EXIT TO MONITOR W/COMMAND

Possible Errors: None

4.3.28 XEXZ - EXIT TO MONITOR W/COMMAND

Mnemonic:	XEXZ
Value:	$A04C
Module:	MPDOSK1
Format:	XEXZ (exits to monitor)
Registers: In	(A1) = Command string

```
EXIT  LEA.L  CMD(PC),A1  ;GET COMMAND
      XEXZ               ;EXIT
*
CMD   DC.B   'PRGM2',0
```

The EXIT TO MONITOR W/COMMAND primitive exits a user program and returns to the PDOS monitor. In addition, the monitor command buffer is loaded with the string pointed to by address register A1. This is useful in passing back parameters to the monitor or to chain to another program.

The exit can be intercepted by changing the value of the EXT$ variable in the task TCB. This primitivie allows you to customize your own monitor.

See also:

4.3.25 XERR - RETURN ERROR D0 TO MONITOR
4.3.27 XEXT - EXIT TO MONITOR

Possible Errors: None

4.3.29 XFAC - FILE ALTERED CHECK

```
    Mnemonic:   XFAC
       Value:   $A0CE
      Module:   MPDOSF
      Format:   XFAC
                  <status error return>

Registers: In   (A1) = FILE NAME
           Out    CC = File not altered
                  CS = File altered
                  NE = Error
```

The FILE ALTERED CHECK primitive looks at the alter bit (bit $80) of the file pointed to by address register A1. If the bit is zero (not altered), then the primitive returns with the carry status bit clear.

If the alter bit is set (file altered), then it is cleared and the primitive returns with carry set. If either case, the bit is always cleared.

Possible Errors: Disk errors

```
XGNP                ;GET PARAMETER
XFAC                ;CHECK FOR FILE ALTERED
  BNE.S @0002       ;ERROR
  BCC.S FALSE       ;NOT ALTERED, RETURN FALSE
BRA.S   TRUE        ;ALTERED, TRUE
```

4.3.30 XFBF - FLUSH BUFFERS

Mnemonic:	XFBF
Value:	$A0F8
Module:	MPDOSF
Format:	XFBF <status error return>
Registers:	None

The FLUSH BUFFERS primitive forces all file slots with active channel buffers to write any updated data to the disk. It thus does a checkpoint of any open and altered file.

Possible Errors: Disk errors

```
LOOP  MOVEQ.L #5*TPS,D0 ;DELAY 5 SECS
      MOVE.W  #128,D1   ;  EVEN 128
      XDEV
      XSUI              ;SUSPEND
      XFBF              ;CHECK POINT DISK
      BRA.S   LOOP
```

4.3.31 XFFN - FIX FILE NAME

```
Mnemonic:    XFFN
   Value:    $A0A0
  Module:    MPDOSF
  Format:    XFFN
               <status error return>

Registers: In  (A1) = File name
          Out  D0.L = Disks(4th/3rd/2nd/1st)
               (A1) = MWB$, Fixed file name
```

```
XGLU           ;GET INPUT LINE
XFFN           ;FIX FILE NAME
  BNE.S ERROR  ;ERROR IN NAME
....
```

The FIX FILE NAME primitive parses a character string for file name, extension, directory level, and disk number. The results are returned in the 32-character monitor work buffer (MWB$(A6)). Data register D0 is also returned with the disk number. The error return is used for an invalid file name.

The monitor work buffer is cleared and the following assignments are made:

```
 0(A1) = File name
 8(A1) = File extension
11(A1) = File directory level
```

```
            0   2   4   6   8   10  12  14  16
            '---'---'---'---'---'---'---'---'...
(A1) ==>    | File name      | Ext |L| 00==>
            '----------------'-----'-'------- ...
```

System defaults are used for the disk number and file directory level when they are not specified in the file name.

See also:

4.3.76 XRDN - READ DIRECTORY ENTRY BY NAME

Possible Errors:

50 = Invalid file name

4.3.32 XFTD - FIX TIME & DATE

```
    Mnemonic:     XFTD
       Value:     $A058
      Module:     MPDOSK3
      Format:     XFTD

Registers: Out    D0.W = Hours * 256 + Minutes
                  D1.W = (Year * 16 + Month) * 32 + Day
```

The FIX TIME & DATE primitive returns a two-word encoded time and date generated from the system timers. The resultant codes include month, day, year, hours, and minutes. The ordinal codes can be sorted and used as inputs to the UNPACK DATE (XUDT) and UNPACK TIME (XUTM) primitives.

Data register D0.W contains the time and register D1.W contains the date. This format is used throughout PDOS for time stamping items.

See also:

- 4.3.57 XPAD - PACK ASCII DATE
- 4.3.77 XRDT - READ DATE
- 4.3.90 XRTM - READ TIME
- 4.3.107 XUAD - UNPACK ASCII DATE
- 4.3.108 XUDT - UNPACK DATE
- 4.3.112 XUTM - UNPACK TIME

Possible Errors: None

```
        LEA.L   TSTP(PC),A0 ;SAVE AREA
        XFTD                ;GET TIME STAMP
        MOVEM.W D0-D1,(A0)  ;SAVE TIME & DATE
        ....

TSTP    DS.W    2           ;TIME STAMP SAVE
```

4.3.33 XFUM - FREE USER MEMORY

```
Mnemonic:      XFUM
   Value:      $A040
  Module:      MPDOSK1
  Format:      XFUM
                 <status error return>

Registers: In  D0.W = Number of K bytes
               (A0) = Beginning address
```

```
MOVEQ.L #20,D0  ;FREE 20K
MOVEA.L A2,A0   ;AT A2
XFUM            ;FREE MEMORY
  BNE.S ERROR
```

The FREE USER MEMORY primitive deallocates user memory to the system memory bit map. Data register D0.W specifies how much memory is to be deallocated while address register A0 points to the beginning of the data block.

Memory thus deallocated is available for any task use including new task creation.

Possible Errors:

79 = Memory error

4.3.34 XGCB - CONDITIONAL GET CHARACTER

```
     Mnemonic:    XGCB
        Value:    $A048
       Module:    MPDOSK2
       Format:    XGCB
                    <status return>

Registers: Out   D0.L = Character in bits 0-7
                   SR = EQ....No character
                        LO....[CTRL-C]
                        LT....[ESC]
                        MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGCB ignores [CTRL-C] and [ESC].

The CONDITIONAL GET CHARACTER primitive checks for a character from first, the input message pointer (IMP$(A6)), second, the assigned input file (ACI$(A6)), and then finally, the interrupt driven input character buffer (PRT$(A6)). If a character is found, it is returned in the right byte of data register D0.L and the rest of the register is cleared.

If there is no input message, no assigned console port character, and the interrupt buffer is empty, the status is returned as 'EQ'.

The status is returned 'LO' and the break flag cleared if the returned character is a [CTRL-C]. The input buffer is also cleared. Thus, all characters entered after the [CTRL-C] and before the XGCB call are dropped.

The status is returned 'LT' and the break flag cleared if the returned character is the [ESC] character.

For all other characters, the status is returned 'HI' and 'GT'. The break flag is not affected.

Possible Errors: None

```
LOOP    XGCB              ;CHARACTER?
          BEQ.S NONE      ;N
          BLO.S QUIT      ;Y, ^C, DONE
          BLT.S NEXT      ;CONTINUE
        CMPI.B  #'0',D0   ;NUMBER?
        ....
```

4.3.35 XGCC - GET CHARACTER CONDITIONAL

```
     Mnemonic:   XGCC
        Value:   $A078
       Module:   MPDOSK2
       Format:   XGCC
                   <status return>

   Registers: Out  D0.L = Character in bits 0-7
                     SR = EQ....No character
                          LO....[CTRL-C]
                          LT....[ESC]
                          MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGCC ignores [CTRL-C] and [ESC].

The GET CHARACTER CONDITIONAL primitive checks the interrupt driven input character buffer and returns the next character in the right byte of data register D0.L. The rest of the register is cleared. The input buffer is selected by the input port variable (PRT$) of the TCB.

If the buffer is empty, the 'EQ' status bit is set. If the character is a [CTRL-C], then the break flag and input buffer are cleared, and the status is returned 'LO'. If the character is the [ESC] character, then the break flag is cleared and the status is returned 'LT'.

If no special character is encountered, the character is returned in register D0 and the status set 'HI' and 'GT'.

If no port has been assigned for input (ie. port 0 or phantom port), then the routine always returns an 'EQ' status.

Possible Errors: None

```
          ....
          XGCC            ;CHARACTER?
            BEQ.S CONT    ;N, CONTINUE
            BLO.S QUIT    ;Y, ^C, QUIT
            BLT.S NEXT    ;Y, ESC, GOTO NEXT
*
WAIT      XGCR            ;Y, WAIT CHARACTER
*
CONT      ....
```

4.3.36 XGCP - GET PORT CHARACTER

```
Mnemonic:      XGCP
   Value:      $A09E
  Module:      MPDOSK2
  Format:      XGCP
                 <status return>

Registers: Out  D0.L = Character in bits 0-7
                  SR = LO....[CTRL-C]
                       LT....[ESC]
                       MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGCP ignores [CTRL-C] and [ESC].

```
LOOP    XGCP              ;GET PORT CHARACTER
          BLO.S QUIT      ;^C, DONE
          BLT.S NEXT      ;CONTINUE
        CMPI.B  #'0',D0   ;NUMBER?
        ....
```

The GET PORT CHARACTER primitive checks for a character in the interrupt driven input character buffer. If a character is found, it is returned in the right byte of data register D0.L and the rest of the register is cleared. The input buffer is selected by the input port variable (PRT$) of the TCB.

If the interrupt buffer is empty, the task is suspended pending a character interrupt.

The status is returned 'LO' and the break flag cleared if the returned character is a [CTRL-C]. The input buffer is also cleared. Thus, all characters entered after the [CTRL-C] and before the XGCR call are dropped.

The status is returned 'LT' and the break flag cleared if the returned character is the [ESC] character.

For all other characters, the status is returned 'HI' and 'GT'. The break flag is not affected.

If no port has been assigned for input, (ie. port 0 or phantom port), then an error 86 occurs.

Possible Errors: None

4.3.37 XGCR - GET CHARACTER

```
Mnemonic:      XGCR
   Value:      $A07A
  Module:      MPDOSK2
  Format:      XGCR
                 <status return>

Registers: Out D0.L = Character in bits 0-7
                 SR = LO....[CTRL-C]
                      LT....[ESC]
                      MI....[CTRL-C] or [ESC]

Note: If the ignore control character bit ($02)
      of the port flag is set, then XGCR ignores
      [CTRL-C] and [ESC].
```

```
LOOP    XGCR                ;GET CHARACTER
          BLO.S QUIT        ;^C, DONE
          BLT.S NEXT        ;CONTINUE
        CMPI.B  #'0',D0     ;NUMBER?
        ....
```

The GET CHARACTER primitive checks for a character from first, the input message pointer (IMP$(A6)); second, the assigned input file (ACI$(A6)); and then finally, the interrupt driven input character buffer (PRT$(A6)). If a character is found, it is returned in the right byte of data register D0.L and the rest of the register is cleared.

If there is no input message, no assigned console port character, and the interrupt buffer is empty, the task is suspended pending a character interrupt.

The status is returned 'LO' and the break flag cleared if the returned character is a [CTRL-C]. The input buffer is also cleared. Thus, all characters entered after the [CTRL-C] and before the XGCR call are dropped.

The status is returned 'LT' and the break flag cleared if the returned character is the [ESC] character.

For all other characters, the status is returned 'HI' and 'GT'. The break flag is not affected.

If no port has been assigned for input, (ie. port 0 or phantom port), then an error 86 occurs.

Possible Errors: None

4.3.38 XGLB - GET LINE IN BUFFER

```
Mnemonic:     XGLB
   Value:     $A07C
  Module:     MPDOSK2
  Format:     XGLB
                {BLT.x ESCAPE}   optional
                <status return>

Registers: In   (A1) = Buffer address
          Out  D1.L = Number of characters
                 SR = EQ...[CR] only
                      LT...[ESC]
                      LO...[CTRL-C]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGLB ignores [CTRL-C] and [ESC].

The GET LINE IN BUFFER primitive gets a character line into the buffer pointed to by address register A1. The XGCR primitive is used by XGLB and hence characters can come from a memory message, a file, or the task console port.

The buffer must be at least 80 characters in length. The line is delimited by a carriage return. The status returns EQUAL if only a [CR] is entered.

If an [ESC] is entered, the task exits to the PDOS monitor unless a 'BLT' instruction immediately follows the XGLB call. If such is the case, then XGLB returns with status set at 'LT'.

If the assigned console flag (ACI$(A6)) is set, then the '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'.

The command line can be edited with various system defined control characters. A [BACKSPACE] ($08) moves the cursor one character to the left. A [CTRL-F] ($0C) moves the cursor one character to the right. A [RUB] ($7F) deletes one character to the left. A [CTRL-D] ($04) deletes the character under the cursor. The cursor need not be at the end of the line when the [CR] is entered.

See also:

4.3.40 XGLU - GET LINE IN USER BUFFER

Possible Errors: None

```
OPEN    XPMC    MES01         ;PROMPT
        LEA.L   BUF(PC),A2    ;GET BUFFER ADDRESS
        XGLB                  ;GET LINE IN BUFFER
          BLT.S OPEN          ;DO NOT EXIT ON ESC
          BEQ.S OPEN10        ;USE DEFAULT
*
OPEN2   XSOP                  ;OPEN FILE
          BNE.S OPEN4         ;ERROR
        ....

OPEN4   CMPI.W  #53,D0        ;'NOT DEFINED' ERROR?
          BNE.S OPERR         ;N
        XDFL                  ;Y, DEFINE FILE, ERROR
          BEQ.S OPEN2         ;N
*
OPERR   XERR                  ;Y, REPORT ERROR
*
OPEN10  ....


MES01   DC.B    $0A,$0D,'FILE=',0
BUF     DS.B    80
```

4.3.39 XGLM - GET LINE IN MONITOR BUFFER

```
  Mnemonic:    XGLM
     Value:    $A07E
    Module:    MPDOSK2
    Format:    XGLM
                 {BLT.x ESCAPE}   optional
                 <status return>

Registers: Out (A1) = String
               D1.L = Number of characters
                 SR = EQ...[CR] only
                      LT...[ESC]
                      LO...[CTRL-C]

Note: If the ignore control character bit ($02)
      of the port flag is set, then XGLM ignores
      [CTRL-C] and [ESC].
```

```
XGLM           ;GET LINE
  BEQ.S NONE
```

The GET LINE IN MONITOR BUFFER primitive gets a character line into the monitor buffer located in the task control block. The XGCR primitive is used by XGLM and hence, characters can come from a memory message, a file, or the task console port.

The buffer has a maximum length of 80 characters and is delimited by a carriage return. The status returns EQUAL if only a [CR] is entered.

If an [ESC] is entered, the task exits to the PDOS monitor unless a 'BLT' instruction immediately follows the XGLM call. If such is the case, then XGLM returns with status set at 'LT'.

If the assigned console flag (ACI$(A6)) is set, then the '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'.

The command line can be edited with various system-defined control characters. A [BACKSPACE] ($08) moves the cursor one character to the left. A [CTRL-L] ($0C) moves the cursor one character to the right. A [RUB] ($7F) deletes one character to the left. A [CTRL-D] ($04) deletes the character under the cursor. The cursor need not be at the end of the line when the [CR] is entered.

The last command line can be recalled to the buffer by entering a [CTRL-A] ($01). This line can then be edited using the above control characters.

Possible Errors: None

4.3.40 XGLU - GET LINE IN USER BUFFER

```
     Mnemonic:   XGLU
        Value:   $A080
       Module:   MPDOSK2
       Format:   XGLU
                   {BLT.x ESCAPE   ;optional}
                   <status return>

Registers: Out   (A1) = String
                 D1.L = Number of characters
                   SR = EQ...[CR] only
                        LT...[ESC]
                        LO...[CTRL-C]

Note: If the ignore control character bit ($02)
      of the port flag is set, then XGLU ignores
      [CTRL-C] and [ESC].
```

The GET LINE IN USER BUFFER primitive gets a character line into the user buffer. Address register A6 normally points to the user buffer. The XGCR primitive is used by XGLU; hence, characters come from a memory message, a file, or the task console port. The line is delimited by a carriage return. The status returns EQUAL if only a [CR] is entered. Address register A1 is returned with a pointer to the first character.

The user buffer is located at the beginning of the task control block and is 256 characters in length. However, the XGLU routine limits the number of input characters to 78 plus two nulls.

If an [ESC] ($1B) is entered, the task exits to the PDOS monitor unless a 'BLT' instruction immediately follows the XGLU call. If such is the case, then XGLU returns with status set at 'LT'.

If the assigned console flag (ACI$(A6)) is set, then the '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'.

The command line can be edited with various system defined control characters. A [BACKSPACE] ($08) moves the cursor one character to the left. A [CTRL-L] ($0C) moves the cursor one character to the right. A [RUB] ($7F) deletes one character to the left. A [CTRL-D] ($04) deletes the character under the cursor. The cursor need not be at the end of the line when the [CR] is entered.

Possible Errors: None

```
GETN   MOVEQ.L #DNUM,D4 ;GET DEFAULT #
       XGLU             ;GET LINE
         BEQ.S GETN2    ;USE DEFAULT
       XCBD             ;CONVERT #, ERROR?
         BLE.S ERROR    ;Y
       MOVE.L  D1,D4    ;N
*
GETN2  MOVE.L  D4,-(A7) ;SAVE #
       ....
```

4.3.41 XGML - GET MEMORY LIMITS

Mnemonic:	XGML
Value:	$A010
Module:	MPDOSK1
Format:	XGML

Registers: Out (A0) = End TCB (TBE$)
(A1) = Upper memory limit (EUM$-USZ)
(A2) = Last loaded address (BUM$)
(A5) = System RAM (SYRAM)
(A6) = Task TCB

```
START   XGML            ;GET MEMORY LIMITS
*
START2  CLR.B   (A2)+   ;CLEAR MEMORY
        CMPA.L  A1,A2   ;DONE?
          BLO.S START2  ;N
        ....
```

The GET MEMORY LIMITS subroutine returns the user task memory limits. These limits are defined as the first usable location after the task control block ($500 beyond address register A6) and the end of the user task memory. The task may use up to but not including the upper memory limit.

Address register A0 is returned pointing to the beginning of user storage (which is the end of the TCB). Register A1 points to the upper task memory limit less $100 hexadecimal bytes for the user stack pointer (USP). Register A2 is the last loaded memory address as provided by the PDOS loader. Address registers A5 and A6 are returned with the pointers to system RAM (SYRAM) and the task control block (TCB).

Possible Errors: None

4.3.42 XGMP - GET MESSAGE POINTER

```
Mnemonic:      XGMP
   Value:      $A004
  Module:      MPDOSK1
  Format:      XGMP
                 <status return>

Registers: In  D0.L = Message slot number (0..15)
          Out  D0.L = Source task # (-1 = no message)
                 SR = EQ....Message (Event[64+Message slot #]=0)
                      NE....No message
               D0.L = Error number 83 if no message
               (A1) = Message
```

The GET MESSAGE POINTER primitive looks for a task message pointer. If no message is ready, then data register D0 returns with a minus one (-1) and status is set to 'Not Equal'.

If a message is waiting, then data register D0 returns with the source task number, address register A1 returns with the message pointer, event (64 + message slot #) is set to zero indicating message received, and status is returned equal.

See also:

- 4.3.44 XGTM - GET TASK MESSAGE
- 4.3.48 XKTM - KILL TASK MESSAGE
- 4.3.96 XSMP - SEND MESSAGE POINTER
- 4.3.99 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message slot empty

4.3.43 XGNP - GET NEXT PARAMETER

```
   Mnemonic:     XGNP
      Value:     $A05A
     Module:     MPDOSM
     Format:     XGNP
                   <status return>

Registers: Out   SR = LO....No parameter
                          [(A1)=0]
                      EQ....Null Parameter
                          [(A1)=0]
                      HI....Parameter
                          [(A1)=PARAMETER]
```

```
SPAC    MOVE.B  SDK$(A6),D0 ;GET SYSTEM DISK #
        XGNP                ;GET PARAMETER, OK?
          BLS.S SPAC02      ;N, USE DEFAULT
        XCDB                ;Y, CONVERT, OK?
          BLE.S ERR67       ;N, ERROR
        MOVE.L  D1,D0       ;Y
*
SPAC02  XSZF                ;GET DISK SIZE
          BNE.S ERROR       ;PROBLEM
        ....
```

The GET NEXT PARAMETER primitive parses the monitor buffer for the next command parameter. The routine does this by maintaining a current pointer into the command line buffer (CLB$) and a parameter delimiter (CMD$).

The XGNP primitive clears all leading spaces of a parameter. A parameter is a character string delimited by a space, comma, period, or null. If a parameter begins with a left parenthesis, then all parsing stops until a matching right parenthesis or null is found. Hence, spaces, commas, and periods are passed in a parameter when enclosed in parentheses. Parentheses may be nested to any depth.

```
x>MASM SOURCE,BIN LIST ERR.SP
x>CT (ASM SOURCE,BIN),15,,3
x>DO ((DO DO),DO)
```

A 'LO' status is returned if the last parameter delimiter is a null or period. XGNP does not parse past a period. In this case, address register A1 is returned pointing to a null string.

```
x>LS.LS
```

An 'EQ' status is returned if the last parameter delimiter is a comma and no parameter follows. Address register A1 is returned pointing to a null string.

```
x>MASM SOURCE,,,ERR
```

A 'HI' status is returned if a valid parameter is found. Address register A1 then points to the parameter.

Possible Errors: None

4.3.44 XGTM - GET TASK MESSAGE

```
    Mnemonic:    XGTM
       Value:    $A01E
      Module:    MPDOSK1
      Format:    XGTM
                   <status return>

  Registers: In  (A1) = Buffer address
             Out D0.L = Source task #
                          (-1 = no message)
                   SR = EQ....message found
                        NE....no message
```

```
LOOP    LEA.L   BUF(PC),A1      ;GET BUFFER ADR
        XGTM                    ;LOOK FOR MESSAGE
          BNE.S NONE            ;NONE
        XPCL                    ;OK, OUT CRLF
        XPLC                    ;OUT MESSAGE
        BRA.S   LOOP            ;LOOK AGAIN
*
NONE    ....

BUFFER  DS.B    64              ;MESSAGE BUFFER
```

The GET TASK MESSAGE primitive searches the PDOS message buffers for a message with a destination equal to the current task number. If a message is found, it is moved to the buffer pointed to by address register A1. The message buffer is then released, and the status is set EQUAL. If no message is found, status is returned NE.

The buffer must be at least 64 bytes in length. (This is a configuration parameter.) The message buffers are serviced on a first in, first out basis (FIFO). Messages are data independent and pass any type of binary data.

See also:

4.3.42 XGMP - GET MESSAGE POINTER
4.3.48 XKTM - KILL TASK MESSAGE
4.3.96 XSMP - SEND MESSAGE POINTER
4.3.99 XSTM - SEND TASK MESSAGE

Possible Errors: None

4.3.45 XGUM - GET USER MEMORY

```
     Mnemonic:   XGUM
        Value:   $A03E
       Module:   MPDOSK1
       Format:   XGUM
                   <status error return>

Registers: In    D0.W = Number of K bytes
          Out    (A0) = Beginning memory address
                 (A1) = End memory address
```

The GET USER MEMORY primitive searches the system memory bit map for a contiguous block of memory equal to D0.W k bytes. If found, the 'EQ' status is set, address registers A0 and A1 are returned the the start and end memory address, and the memory block is marked as allocated in the bit map.

See also:

4.3.33 XFUM - FREE USER MEMORY

Possible Errors:

73 = Not enough memory

```
GETM  CLR.W    -(A7)      ;PUSH .NE.
      MOVEQ.L  #10,D0     ;GET 10K BYTES
      XGUM
        BNE.S  @GM02      ;ERROR
      MOVE.L   A0,AV(A6)  ;SAVE
      ADDQ.W   #$04,(A7)  ;RETURN .EQ.
*
@GM02 RTR                 ;RETURN
```

4.3.46 XISE - INITIALIZE SECTOR

```
Mnemonic:      XISE
   Value:      $A0C0
  Module:      MPDOSF
  Format:      XISE
                 <status error return>

Registers: In  D0.B = Disk number
               D1.W = Logical sector number
               (A2) = Buffer address
```

The INIT SECTOR primitive is a system-defined, hardware-dependent program which writes 256 bytes of data from a buffer (A2) to a logical sector number (D1) on disk (D0). This routine is meant to be used only for disk initialization and is equivalent to the WRITE SECTOR (XWSE) primitive for all sectors except 0. Sector 0 is not checked for the PDOS ID code.

See also:

CHAPTER 8 BIOS
4.3.85 XRSE - READ SECTOR
4.3.88 XRSZ - READ SECTOR ZERO
4.3.118 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

```
        MOVEQ.L DSKN,D0         ;GET DISK #
        MOVEQ.L #0,D1           ;START AT SECTOR
        LEA.L   BUF(PC),A2      ;GET BUFFER PTR
*
LOOP    XISE                    ;WRITE TO DISK
          BNE.S ERROR           ;ERROR
        ADDQ.W  #1,D1           ;MOVE TO NEXT
        CMPI.W  #DISKZ,D1       ;DONE?
          BLO.S LOOP            ;N
        ....
```

4.3.47 XKTB - KILL TASK

Mnemonic: XKTB
Value: $A0FA
Module: MPDOSK1
Format: XKTB
<status error return>

```
PREND   CLR.B   D0        ;KILL SELF
        XKTB              ;CALL CURRENT TASK
          BNE.S ERROR
```

Registers: In D0.B = Task number

Note: If D0.B equals zero, then kill current task. If D0.B is negative, then kill task without allocating task memory to system bit map.

The KILL TASK primitive removes a task from the PDOS task list and optionally returns the task's memory to the system memory bit map. Only the current task or a task spawned by the current task can be killed. Task 0 cannot be killed.

The kill process includes releasing the input port assigned to the task and closing all files associated with the task.

The task number is specified in data register D0.B. If register D0.B equals zero, then the current task is killed and its memory deallocated in the system memory bit map.

If D0=0, then kill self & deallocate memory

If D0.B is positive, then the selected task is killed and its memory deallocated. If D0.B is negative, then task number ABS(D0.B) is killed, but its memory is not deallocated in the memory bit map.

If D0>0, then kill task D0 & deallocate memory

If D0<0, then kill task ABS(D0) & do not deallocate memory

See also:

4.3.19 XCTB - CREATE TASK BLOCK

Possible Errors:

74 = No such task
76 = Task locked

4.3.48 XKTM - KILL TASK MESSAGE

```
 Mnemonic:      XKTM
    Value:      $A028
   Module:      MPDOSK1
   Format:      XKTM
                  <status return>

Registers: In   D0.B = Task #
                (A1) = Buffer address
           Out  D0.L = Source task #
                        (-1 = no message)
                  SR = EQ....message found
                       NE....no message
```

```
LOOP    MOVEQ.L #0,D0    ;SELECT TASK 0
        LEA.L   BF(PC),A1
        XKTM             ;ANY MESSAGE?
          BEQ.S LOOP     ;Y, DO AGAIN
```

The KILL TASK MESSAGE primitive allows you to read (and thus clear) any task's messages from the system message buffers.

See also:

4.3.42 XGMP - GET MESSAGE POINTER
4.3.44 XGTM - GET TASK MESSAGE
4.3.96 XSMP - SEND MESSAGE POINTER
4.3.99 XSTM - SEND TASK MESSAGE

Possible Errors: None

4.3.49 XLDF - LOAD FILE

```
Mnemonic:     XLDF
   Value:     $A0B0
  Module:     MPDOSF
  Format:     XLDF
                <status error return>

Registers: In  D1.B = Execution flag
               (A0) = Start of load memory
               (A1) = End of load memory
               (A3) = File name
          Out  (A0) = EAD$ - Lowest loaded address
               (A1) = BUM$ - Last loaded address
```

Note: If D1.B=0, then XLDF returns to your calling program. If D1.B<>0, then the program is immediately executed.

The LOAD FILE primitive reads and loads 68000 object code into user memory. The file name pointer is passed in address register A3. Registers A0 and A1 specify the memory bounds for the relocatable load. The file must be typed 'OB' or 'SY'.

If data register D1.B is zero, then XLDF returns to the calling program. Otherwise, the loaded program is immediately executed.

The 68000 object should be position-independent section 0 code without any external references or definitions.

A 'SY' file is generated from an 'OB' file by the MSYFL utility. The condensed object is a direct memory image and must be position-independent code.

The XLDF primitive uses long word moves and may move up to three bytes more than contained in an 'SY' file. As such, you must allow for extra space for data moves to an existing program.

Possible Errors:

63 = Illegal object tag
64 = Illegal section
65 = File not loadable
71 = Exceeds task size
73 = Not enough memory
Disk errors

```
XGML                ;GET MEMORY LIMITS
CLR.L   D0          ;RETURN
ADDA.W  #$100,A0    ;ADD DISPLACEMENT
LEA.L   FN(PC),A3   ;GET FILE NAME
XLDF                ;LOAD FILE
  BNE.S ERROR       ;ERROR
```

Legal tags:

```
OT--LABEL--vvvrrrddddddtttt
1Saaaaaaaa              ;ENTRY POINT
2Saaaaaaaa              ;ADDRESS
3dd                     ;SIMPLE DATA BYTE
4dddd                   ;SIMPLE DATA WORD
5dddddddd               ;SIMPLE LONG DATA WORD
6                       ;POP BYTE
7                       ;POP WORD
8                       ;POP LONG WORD
9Snnnnnnnn              ;PUSH VALUE
Dccccdddd               ;STORE MULTIPLE WORD
ESllllllll              ;SECTION LENGTH
Fcc                     ;END OF RECORD/CHECKSUM
```

Illegal tags:

```
ASl<symbol>             ;PUSH SYMBOL
BO                      ;DO OPERATION
CSl<symbol>nnnnnnnn     ;EXTERNAL DEFINITION
```

4.3.50 XLER - LOAD ERROR REGISTER

Mnemonic: XLER
Value: $A03A
Module: MPDOSK1
Format: XLER

Registers: In D0.W = Error number

The LOAD ERROR REGISTER primitive stores data register D0.W in the task control block variable LEN$(A6). This variable will replace the parameter substitution variable '&0' during a procedure file.

User programs should execute this call when an error occurs.

The enable echo flag (ECF$(A6)) is cleared by this call.

Possible Errors: None

```
ADDI.W  #300,D0 ;BIAS ERROR #
XLER            ;REPORT TO PDOS
```

4.3.51 XLFN - LOOK FOR NAME IN FILE SLOTS

```
     Mnemonic:   XLFN
        Value:   $A0A2
       Module:   MPDOSF
       Format:   XLFN
                   &lt;status return&gt;

Registers: In    D0.B = Disk number
                 (A1) = Fixed file name
          Out    D3.W = File ID (Disk #/Index)
                 (A3) = Slot entry address
                   SR = NE...File name not found
                        EQ...File name found

Note: If D3.W=0, then no slots are available.
```

The LOOK FOR NAME IN FILE SLOTS primitive searches through the file slot table for the file name as specified by registers D0.B and A1. If the name is not found, register D3.W returns with a -1 or 0. The latter indicates the file was not found and there are no more slots available. Otherwise, register D3.W returns the associated file ID and register A3 returns the address of the file slot.

A file slot is a 38-byte buffer where the status of an open file is maintained. There are 32 file slots available. The file ID consists of the disk # and the file slot index.

File slots assigned to read-only files are skipped and not considered for file match.

Possible Errors: None

```
XNOP    LEA.L   FN(PC),A1 ;POINT TO FILE NAME
        XFFN              ;FIX FILE NAME
          BNE.S ERR1      ;ERROR
        XLFN              ;LOOKUP NAME, FOUND?
          BEQ.S ERR2      ;Y, FILE ALREADY OPEN
        ....

ERR1    XPMC    MERR1     ;INVALID FILE NAME
        RTS
*
ERR2    XPMC    MERR2     ;FILE ALREADY OPEN
        RTS
*
FN      DC.B    'FILENAME',0
MERR1   DC.B    $0A,$0D,'INVALID FILE NAME',0
MERR2   DC.B    $0A,$0D,'FILE ALREADY OPEN',0
        EVEN
```

```
File slot format:  (38 bytes)

   0(A3) = File name.11
  11(A3) = Level.1
  12(A3) = Status.2
  14(A3) = Sector # in memory.2
  16(A3) = Pointer.4
  20(A3) = Sector index in memory.2
  22(A3) = Sector index of eof.2
  24(A3) = # bytes in end sector.2
  26(A3) = Lock.1/shared flag.1
  28(A3) = Channel buffer ptr.4
  32(A3) = Lock.1/shared flag.1
  34(A3) = Roll-out error #.2
  36(A3) = Disk #.2

Status: 01xx    Sequential
        02xx    Random
        06xx    Shared random
        0Axx    Read only random
        10xx    Driver in channel
        ....
        xx80    Altered
        xx04    Contiguous
        xx02    Delete protect
        xx01    Write protect

        8xxx    Sector altered
        4xxx    File altered
        2xxx    Buffer locked in memory
```

4.3.52 XLKF - LOCK FILE

```
Mnemonic:      XLKF
   Value:      $A0D8
  Module:      MPDOSF
  Format:      XLKF
                 <status error return>

Registers: In  D1.W = File ID
```

```
MOVE.W  D5,D1    ;GET FILE ID
XLKF             ;LOCK FILE
  BNE.S ERROR    ;PROBLEM
....
```

The LOCK FILE primitive locks an opened file so that no other task can gain access until an UNLOCK FILE (XULF) primitive is executed. Only the locking task has access to the locked file.

A locked file is indicated by a -1 ($FF) in the left byte of the lock file parameter (LF) of the file slot usage (FS) command. The locking task number is stored in the left byte of the task number parameter (TN).

See also:

4.3.109 XULF - UNLOCK FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
75 = File locked
Disk errors

4.3.53 XLKT - LOCK TASK

```
   Mnemonic:   XLKT
      Value:   $A014
     Module:   MPDOSK1
     Format:   XLKT
                 <status return>

Registers: Out  SR = EQ...Not locked
                     NE...Locked
```

```
          XLKT               ;LOCK TASK
            SNE.B D7         ;SET FLAG
          TAS.B   SBIT       ;START CRITICAL PROCESS
*
WAIT      TST.B   SBIT       ;OK?
            BMI.S WAIT       ;N
          TST.B   D7         ;Y, LEAVE LOCKED?
            BNE.S CONT       ;Y
          XULT               ;N, UNLOCK TASK
*
CONT      ....
```

The LOCK TASK primitive locks the requesting task in the run state by setting the swap lock variable in system RAM to nonzero. The task remains locked until an UNLOCK TASK (XULT) is executed. The status of the lock variable BEFORE the call is returned in the status register.

XLKT waits until all locks (Level 2 and Level 3 locks) are cleared before the task is locked.

See also:

4.3.110 XULT - UNLOCK TASK

Possible Errors: None

4.3.54 XLSR - LOAD STATUS REGISTER

Mnemonic: XLSR
Value: $A02E
Module: MPDOSK1
Format: XLSR

Registers: In D1.W = 68000 status register

The LOAD STATUS REGISTER primitive allows you to directly load the 68000 status register. Of course, only appropriate bits (i.e. the interrupt mask too high, supervisor mode, trace mode, etc.) are to be set so that the system is not crashed.

See also:

4.3.102 XSUP - ENTER SUPERVISOR MODE

Possible Errors: None

```
MOVE.W  SR,D1       ;READ STATUS
ORI.W   #$2000,D1   ;ADD SUPERVISOR
XLSR                ;LOAD SR
```

4.3.55 XLST - LIST FILE DIRECTORY

```
   Mnemonic:   XLST
      Value:   $A0A4
     Module:   MPDOSM
     Format:   XLST
                 <status error return>

Registers: In  (A1) = List specifications
```

```
MLST    XGNP             ;GET SELECT LIST
        XLST             ;CALL FOR LIST
          BNE.S ERROR    ;ERROR
        XEXT             ;EXIT TO MONITOR
```

The LIST FILE DIRECTORY subroutine causes PDOS to output a formatted file directory listing to the console terminal, according to the select string pointed to by address register A1. The output may be interrupted at any time by a character being entered on the console port. An [ESC] character returns control to the PDOS monitor.

The format of the list specifications is defined as follows:

```
DC.B    '{file}{:ext}{;level}{/disk}{/select...}',0

where:  {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
        {:ext} = 1 to 3 characters (:@=all,*=wild)
      {;level} = directory level (;@=all)
       {/disk} = disk number ranging from 0 to 255
     {/select} = /AC = Assign Console file
                 /BN = Binary file
                 /BX = PDOS BASIC token file
                 /EX = PDOS BASIC file
                 /OB = 68000 PDOS object file
                 /SY = System file
                 /TX = Text file
                 /DR = System I/O driver
                 /*  = Delete protected
                 /** = Delete and write protected
                 /Fdy-mon-yr = selects files with date of
                               last change greater than
                               or equal to 'dy-mon-yr'
                 /Tdy-mon-yr = selects files with date of
                               last change less than or
                               equal to 'dy-mon-yr'
```

Possible Errors: Disk Errors

4.3.56 XNOP - OPEN SHARED RANDOM FILE

```
Mnemonic:   XNOP
   Value:   $A0DA
  Module:   MPDOSF
  Format:   XNOP
              <status error return>

Registers: In   (A1) = File name
           Out  D0.W = File attribute
                D1.W = File ID

   Notes: Uses multiple directory file search.

          You MUST lock and position file before
          each multi-task access.
```

```
        LEA.L   FN(PC),A1 ;POINT TO NAME
        XNOP              ;OPEN SHARED
          BNE.S ERROR
        MOVE.W  D0,D5     ;SAVE TYPE
        SWAP    D5
        MOVE.W  D1,D5     ;SAVE FILE ID
        ....

FN      DC.B    'FILENAME:EXT',0
        EVEN
```

The OPEN SHARED RANDOM FILE primitive opens a file for shared random access by assigning the file to an area of system memory called a file slot. The file ID and file attribute are returned to the calling program in registers D1 and D0, respectively. Thereafter, the file is referenced by the file ID and not by the file name. A new entry in the file slot table is made only if the file is not already opened for shared access.

The file ID (returned in register D1) is a 2-byte number. The left byte is the disk number and the right byte is the file slot index. The file attributes are returned in register D0.

```
D0.W = (ABOS BETU xxxx xCWD)
D1.W = (Disk #) x 256 + (file slot index)
```

The END-OF-FILE marker on a shared file is changed only when the file has been extended. All data transfers are buffered through a channel buffer; data movement to and from the disk is by full sectors.

An "opened count" is incremented each time the file is shared-opened and is decremented by each close operation. The file is only closed by PDOS when the count is zero. This count is saved in the right byte of the locked file parameter (LF) and is listed by the file slot usage command (FS).

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors

4.3.57 XPAD - PACK ASCII DATE

```
   Mnemonic:     XPAD
      Value:     $A00A
     Module:  MPDOSK3
     Format:  XPAD

  Registers:  In   (A1) = 'DY-MON-YR'
              Out  D1.W = (Year*16+month)*32+day
                          (YYYY YYYM MMMD DDDD)
                   (A1) = Updated
                     SR = .EQ. - Conversion ok
                          .NE. - Error
```

The PACK ASCII DATE primitive converts an ASCII date string to an encoded binary number in data register D1. The result is compatible with other PDOS date primitives such as XUAD.

See Also:

4.3.22 XFTD - FIX TIME & DATE
4.3.77 XRDT - READ DATE
4.3.90 XRTM - READ TIME
4.3.107 XUAD - UNPACK ASCII DATE
4.3.108 XUDT - UNPACK DATE

Possible Errors: Status errors.

```
STRT XPMC    MES1  ;DATE=
     XGLU          ;GET LINE
     XPAD          ;CONVERT
       BNE.S ERR   ;ERROR
     XPMC    MES2  ;D1.W=
     XCBH
     ADDQ.W  #4,41
     XPLC          ;OUTPUT
     BRA.S   STRT
*
ERR  XPMC    MES3  ;ERROR
     BRA.S   STRT
*
MES1 DC.B $0A,$0D,'DATE=',0
MES2 DC.B '  D1.W=$',0
MES3 DC.B $0A,$0D,'*ERROR',0
     EVEN
     END     STRT

x>TEST
DATE=11-NOV-86  D1.W=$AD6B
DATE=11NOV86  D1.W=$AD6B
DATE= NOV 11 86
*ERROR
DATE=
```

4.3.58 XPBC - PUT BUFFER TO CONSOLE

Mnemonic:	XPBC
Value:	$A084
Module:	MPDOSK2
Format:	XPBC
Registers:	None

The PUT USER BUFFER TO CONSOLE primitive outputs the ASCII contents of the user buffer to the user console and/or SPOOL file. The output string is delimited by the null character. The user buffer is the first 256 bytes of the task control block and is pointed to by address register A6.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.38 XGLB - GET LINE IN BUFFER

Possible Errors: None

```
CLINE   MOVEA.L A6,A2      ;GET USER BUFFER PTR
*
CLINE2
        ....

        MOVE.B  D0,(A2)+ ;LOAD BUFFER, DONE?
          BNE.S CLINE2   ;N
        XPBC             ;Y, OUTPUT BUFFER
        RTS              ;CONTINUE
```

4.3.59 XPCB - PUSH COMMAND TO BUFFER

Mnemonic:	XPCB
Value:	$A04E
Module:	MPDOSM
Format:	XPCB

Registers: In (A1) = Command string

The PUSH COMMAND TO BUFFER primitive pushes the string pointed to by address register A1 into the command recall buffer. Since there is a limit on the buffer size, older commands are lost.

See also:

4.3.43 XGNP - GET NEXT PARAMETER

Possible Errors: None

```
        XGLU            ;GET COMMAND
        XPCB            ;PUSH FOR RECALL
        ...
```

4.3.60 XPCC - PUT CHARACTER(S) TO CONSOLE

Mnemonic:	XPCC
Value:	$A086
Module:	MPDOSK2
Format:	XPCC

Registers: In DO.W = Character(s)

The PUT CHARACTER TO CONSOLE primitive outputs one or two ASCII characters in data register DO to the user console and/or SPOOL file. The right byte (bits 0 through 7) is first and is followed by the left byte (bits 8 through 15) if non-zero. If the right byte or both bytes are zero, nothing is output to the console.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.62 XPCR - PUT CHARACTER RAW
4.3.63 XPDC - PUT DATA TO CONSOLE

possible Errors: None

```
MOVE.W  #'C^',DO  ;OUTPUT '^C'
XPCC
MOVEQ.L #$0A,DO   ;FOLLOWED BY LF
XPCC
```

4.3.61 XPCL - PUT CRLF TO CONSOLE

Mnemonic: XPCL
Value: $A088
Module: MPDOSK2
Format: XPCL

```
XPCL            ;OUTPUT CRLF
...
```

Registers: None

The PUT CRLF TO CONSOLE primitive outputs the ASCII characters carriage return <$0A> and line feed <$0D> to the user console and/or SPOOL file. The column counter is cleared.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

Possible Errors: None

4.3.62 XPCR - PUT CHARACTER RAW

Mnemonic:	XPCR
Value:	$AOBA
Module:	MPDOSK2
Format:	XPCR

Registers: In DO.B = CHARACTER

The PUT CHARACTER RAW primitive outputs the character in the lower byte of data register DO to the user console. No attempt is made by PDOS to interpret control characters.

See also:

4.3.60 XPCC - PUT CHARACTER(S) TO CONSOLE
4.3.63 XPDC - PUT DATA TO CONSOLE

Possible Errors: None

4.3.63 XPDC - PUT DATA TO CONSOLE

Mnemonic:	XPDC
Value:	$A096
Module:	MPDOSK2
Format:	XPDC
Registers: In	D7.W = LENGTH
	(A1) = DATA STRING

```
          MOVEQ.L #0,D7
          LEA.L   M(PC),A1 ;POINT TO STRING
          MOVE.B  (A1)+,D7 ;GET LENGTH
          XPDC             ;OUTPUT
          ...

M         DC.B    10,$0A,$0D
          DC.B    'THIS IS A MESSAGE'
```

The PUT DATA TO CONSOLE primitive outputs data-independent bytes to the console. Address register A1 points to the string while data register D7 has the string length.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.60 XPCC - PUT CHARACTER(S) TO CONSOLE
4.3.62 XPCR - PUT CHARACTER RAW

Possible Errors: None

4.3.64 XPEL - PUT ENCODED LINE TO CONSOLE

```
Mnemonic:   XPEL
   Value:   $A06E
  Module:   MPDOSK2
  Format:   XPEL

Registers: In  (A1) = Message
```

The PUT ENCODED LINE TO CONSOLE primitive outputs to the user console the message pointed to by address register A1. An encoded message is similar to any other string with the exception that the parity bit is used to output blanks and the character $80 outputs a carriage return/line feed.

If the parity bit is set and the masked character ($7F) is less than or equal to a blank, then the numeric value of the negated character is used as the number of blanks to be inserted in the output stream. If the mask character is greater than a blank, then that character is output followed by one blank.

With the exception of control characters, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.65 XPEM - PUT ENCODED MESSAGE TO CONSOLE
4.3.66 XPLC - PUT LINE TO CONSOLE
4.3.67 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

```
         LEA.L   M(PC),A1 ;POINT TO MESSAGE
         XPEL              ;OUTPUT MESSAGE
         ....

M        DC.B    $80,'Lev',-2,'Name:ext'
         DC.B    -6,'Type',-6,'Size',-6
         DC.B    'Dat',-'e','created',-4
         DC.B    'Las',-'t','update',0
```

Note: The above text strings are equivalent to:

```
M        DCE.B   $80,'Lev  Name:ext'
         DCE.B   '      Type      Size'
         DCE.B   '      Date created'
         DCE.B   '    Last update',0
```

$80 = Carriage return/line feed

4.3.65 XPEM - PUT ENCODED MESSAGE TO CONSOLE

```
Mnemonic:   XPEM
   Value:   $A09C
  Module:   MPDOSK2
  Format:   XPEM    <message>

Registers:  None
```

The PUT ENCODED MESSAGE TO CONSOLE primitive outputs to the user console the PC relative message contained in the word following the call. An encoded message is similar to any other string with the exception that the parity bit is used to output blanks and the character $80 outputs a carriage return/line feed.

If the parity bit is set and the masked character ($7F) is less than or equal to a blank, then the numeric value of the negated character is used as the number of blanks to be inserted in the output stream. If the mask character is greater than a blank, then that character is output followed by one blank.

With the exception of control characters, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.64 XPEL - PUT ENCODED LINE TO CONSOLE
4.3.66 XPLC - PUT LINE TO CONSOLE
4.3.67 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

```
         XPEM    MES01     ;OUTPUT MESSAGE
         ....

MES01    DC.B    $80,'Lev',-2,'Name:ext'
         DC.B    -6,'Type',-6,'Size',-6
         DC.B    'Dat',-'e','created',-4
         DC.B    'Las',-'t','update',0
```

$80 = Carriage return/line feed

4.3.66 XPLC - PUT LINE TO CONSOLE

Mnemonic:	XPLC
Value:	$A08A
Module:	MPDOSK2
Format:	XPLC

Registers: In (A1) = ASCII string

The PUT LINE TO CONSOLE primitive outputs the ASCII character string pointed to by address register A1 to the user console and/or SPOOL file. The string is delimited by the null character.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.64 XPEL - PUT ENCODED LINE TO CONSOLE
4.3.65 XPEM - PUT ENCODED MESSAGE TO CONSOLE
4.3.67 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

```
         LEA.L   MES1(PC),A1      ;OUTPUT MESSAGE
         XPLC
         MOVE.L  NUMB(PC),D1      ;GET NUMBER
         XCBD                     ;CONVERT TO DECIM
         XPLC                     ;OUTPUT
         ....

NUMB     DS.L    1                ;NUMBER HOLDER
MES1     DC.B    $0A,$0D          ;MESSAGE #1
         DC.B    'ANSWER=',0
```

4.3.67 XPMC - PUT MESSAGE TO CONSOLE

Mnemonic:	XPMC
Value:	$A08C
Module:	MPDOSK2
Format:	XPMC <message>
Registers:	None

```
        XPMC    MES2    ;OUTPUT HEADER
        ....

MES2    DC.B    $0A,$0D ;HEADER MESSAGE
        DC.B    'PDOS REV 3.0',0
```

The PUT MESSAGE TO CONSOLE primitive outputs the ASCII character string pointed to by the message address word immediately following the PDOS call to the user console and/or SPOOL file. The address is a PC relative 16-bit displacement to the message. The output string is delimited by the null character.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.64 XPEL - PUT ENCODED LINE TO CONSOLE
4.3.65 XPEM - PUT ENCODED MESSAGE TO CONSOLE
4.3.66 XPLC - PUT LINE TO CONSOLE

Possible Errors: None

4.3.68 XPSC - POSITION CURSOR

```
Mnemonic:      XPSC
   Value:      $A08E
  Module:      MPDOSK2
  Format:      XPSC

Registers: In  D1.B = Row
               D2.B = Column
```

Note: Uses PSC$(A6) as lead characters.

```
OUTM   MOVEQ.L #23,D1   ;POSITION TO BOTTOM
       CLR.L   D2       ;  OF SCREEN
       XPSC             ;POSITION
       XPMC    MES1     ;OUTPUT MESSAGE
       ....
```

The POSITION CURSOR primitive positions the cursor on the console terminal according to the row and column values in data registers D1 and D2. Register D1 specifies the row on the terminal and generally ranges from 0 to 23, with 0 being the top row. Register D2 specifies the column of the terminal and ranges from 0 to 79, with 0 being the left-hand column. Register D2 is also loaded into the column counter reflecting the true column of the cursor.

The XPSC primitive outputs either one or two leading characters followed by the row and column. The leading characters output by XPSC are located in PSC$(A6) of the task control block. These characters are transferred from the parent task to the spawned task during creation. The initial characters come from the BIOS module.

```
CSC$(A6) = E111 1111 E222 2222
           \\       \ \\     \_
            \\       \ \\_______ 2nd character
             \\       \ \_______ 2nd [ESC]
              \\       \
               \\       \_______
                \\______________ 1st character
                 \______________ 1st [ESC]|
```

The row and column characters are biased by $20 if the parity bit of the first character is set. Likewise, if the second character's parity bit is set, then row/column order is reversed. This accommodates most terminal requirements for positioning the cursor.

If PSC$ is zero, then PDOS makes a call into the BIOS for custom position cursor. The entry point is B_PSC beyond the BIOS table.

```
MOVE.W  PSC$(A6),(A3)   ;GET CHARACTERS, BIAS?
  BLT.S @0002           ;Y
  BGT.S @0004           ;N
MOVEA.L (A5),A0         ;N, POINT TO BIOS
JSR     B_PSC(A0)       ;GET BIOS POSITION CODE
  BNE.S XCLS10          ;TERMINATE AND OUTPUT
```

The MTERM utility is used to change the position cursor codes.

See also:

4.3.17 XCLS - CLEAR SCREEN
4.3.73 XRCP - READ PORT CURSOR POSITION
CHAPTER 8 - BIOS

Possible Errors: None

4.3.69 XPSF - POSITION FILE

Mnemonic: XPSF
Value: $A0DC
Module: MPDOSF
Format: XPSF
<status error return>

Registers: In D1.W = File ID
D2.L = Byte position

Note: A byte position equal to -1 positions to the end of the file.

The POSITION FILE primitive moves the file byte pointer to any byte position within a file. The file ID is given in register D1 and the long word byte position is specified in register D2.

An error occurs if the byte position is greater than the current end-of-file marker.

A contiguous file greatly enhances the speed of the position primitive since the desired sector is directly computed. However, the position primitive does work with non-contiguous files, as PDOS follows the sector links to the desired byte position.

A contiguous file is extended by positioning to the end-of-file marker and writing data. However, PDOS will alter the file type to non-contiguous if a contiguous sector is not available. This would result in random access being much slower.

See also:

4.3.79 XRFP - READ FILE POSITION
4.3.93 XRWF - REWIND FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
70 = Position error
Disk errors

```
         MOVE.W  D5,D1      ;GET FILE ID
         MOVE.W  RN(A0),D2  ;GET RECORD #
         MULU.W  #36,D2     ;GET BYTE INDEX
         XPSF               ;POSITION WITHIN FILE
           BNE.S ERROR
         ....

RN       DS.W    1          ;RECORD #
```

4.3.70 XPSP - PUT SPACE TO CONSOLE

Mnemonic:	XPSP
Value:	$A098
Module:	MPDOSK2
Format:	XPSP
Registers:	None

```
MOVEQ.L #N,D1    ;GET NUMBER
XCBM    MES01    ;CONVERT
XPLC             ;OUTPUT LINE
XPSP             ;OUT SPACE
```

The PUT SPACE TO CONSOLE outputs a [SP] ($20) character to the user console. There are no registers or status involved.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:

4.3.60 XPCC - PUT CHARACTER(S) TO CONSOLE

Possible Errors: None

4.3.71 XRBF - READ BYTES FROM FILE

```
     Mnemonic:   XRBF
        Value:   $A0DE
       Module:   MPDOSF
       Format:   XRBF
                   &lt;status error return&gt;

Registers: In    D0.L = Number of bytes
                 D1.W = File ID
                 (A2) = R/W buffer address
            Out  D3.L = Number of bytes read
                        (On EOF only.)
```

The READ BYTES FROM FILE primitive reads the number of bytes specified in register D0 from the file specified by the file ID in register D1 into a memory buffer pointed to by address register A2. If the channel buffer has been rolled to disk, the least-used buffer is freed and the desired buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

If an error occurs during the read operation, the error return is taken with the error number in register D0 and the number of bytes actually read in register D3.

The read is independent of the data content. The buffer pointer in register A2 is on any byte boundary. The buffer is not terminated with a null.

A byte count of zero in register D0 results in one byte being read from the file. This facilitates single byte data acquisition.

See also:

4.3.80 XRLF - READ LINE FROM FILE
4.3.113 XWBF - WRITE BYTES TO FILE
4.3.117 XWLF - WRITE LINE TO FILE

Possible Errors:

52 = File not open
56 = End of file
59 = Invalid slot #
Disk errors

```
        MOVE.L  #256,D0 ;READ 256 BYTES
        MOVE.W  D5,D1   ;GET FILE ID
        MOVEA.L A6,A2   ;READ INTO USER BUF
        XRBF            ;READ DATA
          BNE.S ERROR
        ....

ERROR   CMPI.W  #56,D0  ;EOF?
          BNE.S ERROR2  ;N
        MOVE.L  D3,D0   ;Y, GET # OF BYTES READ
        ....
```

4.3.72 XRCN - RESET CONSOLE INPUTS

Mnemonic:	XRCN
Value:	$A0B2
Module:	MPDOSF
Format:	XRCN
Registers:	None

```
DONE    XRCN            ;CLOSE FILES
```

The RESET CONSOLE INPUTS closes the current procedure file. If there are other procedure files pending (nested), then they become active again.

See also:

4.3.6 XCBC - CHECK FOR BREAK CHARACTER

Possible Errors: None

4.3.73 XRCP - READ PORT CURSOR POSITION

```
  Mnemonic:     XRCP
     Value:     $A092
    Module:     MPDOSK2
    Format:     XRCP

Registers: In   D0.W = Port #
          Out   D1.L = Row
                D2.L = Column
```

Note: If D0.W=0, then the current port (PRT$(A6)) is used.

```
MOVEQ.L #1,D0    ;LOOK AT PORT 1
XRCP             ;READ POSITION
SWAP    D1
MOVE.W  D2,D1    ;D1.L=X/Y POSITION
```

The READ PORT CURSOR POSITION primitive reads the current cursor position for the port designated by data register D0.B. The PDOS system maintains a column count (0-79) and a row count (0-23) for each port. When the cursor reaches row 23, the count is not incremented, acting like a screen scroll.

See also:

4.3.17 XCLS - CLEAR SCREEN
4.3.68 XPSC - POSITION CURSOR

Possible Errors: None

4.3.74 XRDE - READ NEXT DIRECTORY ENTRY

```
   Mnemonic:    XRDE
      Value:    $A0A6
     Module:    MPDOSF
     Format:    XRDE
                  <status error return>

Registers: In   D0.B = Disk number
                D1.B = Read flag (0=1st)
                (A2) = Last 32 byte directory entry
                TW1$ = Sector number
                TW2$ = number of directory entries
          Out   D1.W = Sector number
                (A2) = Next entry
```

```
START   MOVEQ.L #0,D1       ;BEGIN WITH 1ST ENTRY
        BRA.S   LOOP02
*
LOOP    MOVEQ.L #-1,D1      ;READ NEXT ENTRY
*
LOOP02  MOVE.W  D5,R0       ;GET DISK #
        XRDE                ;READ DIRECTORY ENTRY
          BNE.S ERROR       ;ERROR
        MOVE.B  12(A2),R4   ;GET FILE TYPE
        ....
```

The READ NEXT DIRECTORY ENTRY primitive reads sequentially through a disk directory. If register D1.B is zero, then the routine begins with the first directory entry. If register D1.B is nonzero, then based on the last directory entry (pointed to by register A2), the next entry is read.

The calling routine must maintain registers D0.B and A2, the user I/O buffer, and temporary variables TW1$ and TW2$ of the task control block between calls to XRDE.

Possible Errors:

53 = File not defined (End of directory)
68 = Not PDOS disk
Disk errors

4.3.75 XRDM - DUMP REGISTERS

Mnemonic:	XRDM
Value:	$A02A
Module:	MPDOSK1
Format:	XRDM

Registers: In All

The DUMP REGISTERS primitive formats and outputs all the current register values of the 68000 to the user console along with the program counter, status register, and the supervisor stack.

The registers and status are not affected by this primitive.

See also:

4.3.5 XBUG - DEBUG CALL
4.3.23 XDMP - DUMP MEMORY FROM STACK
PB - PDOS DEBUGGER (chapter 3)

Possible Errors: None

```
MOVEM.L RL,(A7)+ ;RESTORE REGISTERS
XRDM             ;DUMP RESULTS
```

4.3.76 XRDN - READ DIRECTORY ENTRY BY NAME

```
    Mnemonic:    XRDN
       Value:    $A0A8
      Module:    MPDOSF
      Format:    XRDN
                   <status error return>

Registers: In    D0.B = Disk number
                 MWB$ = File name
          Out    D1.W = Sector number in memory
                 (A2) = Directory entry
                 TW2$ = Entry count
```

```
OPENF   LEA.L   FN(PC),A1 ;GET FILE NAME POINTER
        XFFN              ;FIX NAME IN MWB
          BNE.S ERROR     ;ERROR
        XRDN              ;READ DIRECTORY ENTRY
          BNE.S ERROR     ;ERROR
        ....
```

The READ DIRECTORY ENTRY BY NAME primitive reads directory entries by file name. Register D0.B specifies the disk number. The file name is located in the Monitor Work Buffer (MWB$) in a fixed format. Several other parameters are returned in the monitor TEMP storage of the user task control block. These variables assist in the housekeeping operations on the disk directory.

See also:

4.3.28 XFFN - FIX FILE NAME

Possible Errors:

53 = File not defined
68 = Not PDOS disk
Disk errors

4.3.77 XRDT - READ DATE

Mnemonic: XRDT
Value: $A05C
Module: MPDOSK3
Format: XRDT

Registers: Out (A1) = 'MN/DY/YR'<null>

The READ DATE primitive returns the current system date as a nine character string. The format is 'MN/DY/YR' followed by a null. Address register A1 points to the string in the monitor work buffer.

See also:

4.3.32 XFTD - FIX TIME & DATE
4.3.57 XPAD - PACK ASCII DATE
4.3.90 XRTM - READ TIME
4.3.107 XUAD - UNPACK ASCII DATE
4.3.108 XUDT - UNPACK DATE
4.3.112 XUTM - UNPACK TIME

Possible Errors: None

```
GETD    XPMC    MES1    ;OUTPUT PROMPT
        XRDT            ;GET DATE
        XPLC            ;OUTPUT TO SCREEN
        ....

MES1    DC.B    'DATE=',0
```

4.3.78 XRFA - READ FILE ATTRIBUTES

```
Mnemonic:     XRFA
   Value:     $A0E0
  Module:     MPDOSF
  Format:     XRFA
                <status error return>

Registers: In (A1) = File name
          Out (A2) = Directory entry
              D0.L = Disk number
              D1.L = File size (in bytes)
              D2.L = Level/attributes

      Note: Uses multiple directory file search.
```

```
        LEA.L   FN(PC),A1 ;GET FILE NAME
        XRFA              ;READ FILE ATTRIBUTES
          BNE.S ERROR     ;PROBLEM
        LRL.W   #2,D2     ;BINARY FILE?
          BCC.S PNO       ;N
        ....              ;Y

FN      DC.B    'PRGM:BIN',0
        EVEN
```

The READ FILE ATTRIBUTES primitive returns the disk number of where the file was found in data register D0.L. Data register D1.L is returned with the size of the file in bytes. The file directory level is returned in the upper word of register D2.L and the file attributes are returned in register D2.W. The file name is pointed to by address register A1. File attributes are defined as follows:

```
$80xx   AC - Procedure file
$40xx   BN - Binary file
$20xx   OB - 68000 object file
$10xx   SY - 68000 memory image
$08xx   BX - BASIC binary token file
$04xx   EX - BASIC ASCII file
$02xx   TX - Text file
$01xx   DR - System I/O driver

$xx04   C  - Contiguous file
$xx02   *  - Delete protect
$xx01   ** - Delete and write protect
```

See also:

4.3.13 XCFA - CLOSE FILE W/ATTRIBUTE
4.3.115 XWFA - WRITE FILE ATTRIBUTES
4.3.116 XWFP - WRITE FILE PARAMETERS

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
Disk errors

4.3.79 XRFP - READ FILE POSITION

```
 Mnemonic:    XRFP
    Value:    $A0FE
   Module:    MPDOSF
   Format:    XRFP
                &lt;status error return>

Registers: In  D1.W = File ID
           Out (A3) = File slot address
               D2.L = Byte position
               D3.L = EOF byte position
```

```
MOVE.W  D5,D1       ;GET FILE ID
XRFP                ;READ FILE POSITION
  BNE.S ERROR
....
```

The READ FILE POSITION primitive returns the current file position, end-of-file position, and file slot address. The open file is selected by the file ID in data register D1.W.

Address register A3 is returned pointing to the open file slot. Data registers D2.L and D3.L are returned with the current file byte position and the end-of-file position respectively.

See also:

4.3.69 XPSF - POSITION FILE
4.3.93 XRWF - REWIND FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
Disk errors

4.3.80 XRLF - READ LINE FROM FILE

```
Mnemonic:     XRLF
   Value:     $A0E2
  Module:     MPDOSF
  Format:     XRLF
                <status error return>

Registers: In  D1.W = File ID
               (A2) = R/W buffer address
           Out D3.L = # of bytes read
                      (On EOF only.)
```

```
        MOVE.W  D5,D1        ;GET FILE ID
        LEA.L   BF(PC),A2    ;GET BUFFER POINTER
        XRLF                 ;READ LINE
          BNE.S ERROR
        ....

BF      DS.B    132          ;MAXIMUM BUFFER NEEDED
```

The READ LINE primitive reads one line, delimited by a carriage return [CR], from the file specified by the file ID in register D1. If a [CR] is not encountered after 132 characters, then the line and primitive are terminated. Address register A2 points to the buffer in user memory where the line is to be stored. If the channel buffer has been rolled to disk, the least-used buffer is freed and the buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

If an error occurs during the read operation, the error return is taken with the error number in register D0 and the number of bytes actually read in register D3.

The line read is dependent upon the data content. All line feeds ([LF]) are dropped from the data stream and the [CR] is replaced with a null. The buffer pointer in register A2 may be on any byte boundary. The buffer is not terminated with a null on an error return.

See also:

4.3.71 XRBF - READ BYTES FROM FILE
4.3.113 XWBF - WRITE BYTES TO FILE
4.3.117 XWLF - WRITE LINE TO FILE

Possible Errors:

52 = File not open
56 = End of file
59 = Invalid slot #
Disk errors

4.3.81 XRNF - RENAME FILE

```
Mnemonic:      XRNF
   Value:      $A0E4
  Module:      MPDOSF
  Format:      XRNF
                 <status error return>

Registers: In  (A1) = Old file name
               (A2) = New file name
```

The RENAME FILE primitive renames a file in a PDOS disk directory. The old file name is pointed to by address register A1. The new file name is pointed to by address register A2.

The XRNF primitive is used to change the directory level for any file by letting the new file name be a numeric string equivalent to the new directory level. XRNF first attempts a conversion on the second parameter before renaming the file. If the string converts to a number without error, then only the level of the file is changed.

See also:

4.3.21 XDFL - DEFINE FILE
4.3.22 XDLF - DELETE FILE

Possible Errors:

50 = Invalid file name
51 = File already defined
Disk errors

```
        LEA.L   F1(PC),A1 ;GET OLD FILE NAME
        LEA.L   F2(PC),A2 ;GET NEW FILE NAME
        XRNF              ;RENAME FILE
          BNE.S ERROR     ;PROBLEM
        MOVEA.L A2,A1     ;POINT TO NEW NAME
        LEA.L   LV(PC),A2 ;GET NEW LEVEL
        XRNF              ;CHANGE DIRECTORY LEVEL
          BNE.S ERROR
        ....

LV      DC.B    '10',0
F1      DC.B    'OBJECT:OLD',0
F2      DC.B    'OBJECT:NEW',0
        EVEN
```

4.3.82 XROO - OPEN RANDOM READ ONLY FILE

```
Mnemonic:     XROO
   Value:     $A0E6
  Module:     MPDOSF
  Format:     XROO
                <status error return>

Registers: In  (A1) = File name
          Out  D0.W = File attribute
               D1.W = File ID

      Note: Uses multiple directory file search.
```

```
        LEA.L   HLPFN(PC),A1 ;POINT TO FILE NAME
        XROO                 ;OPEN FILE
          BNE.S ERROR
*
HELP02  MOVEA.L A6,A2        ;GET BUFFER
        XRLF                 ;READ LINE
          BNE.S SHWF22
        ....

HLPFN   DC.B    'HLPTX',0
```

The OPEN RANDOM READ ONLY FILE primitive opens a file for random access by assigning the file to an area of system memory called a file slot, and returning a file ID and file attribute to the calling program. Thereafter, the file is referenced by the file ID and not by the file name. This type of file open provides read only access.

The file ID (returned in register R1) is a 2-byte number. The left byte is the disk number and the right byte is the channel buffer index. The file attribute is returned in register D0.

```
D1.W = (Disk #) x 256 + (File slot index)
D0.W = (ABOS BETD xxxx xCWD)
```

Since the file cannot be altered, it cannot be extended nor is the LAST UPDATE parameter changed when it is closed. All data transfers are buffered through a channel buffer and data movement to and from the disk is by full sectors.

A new file slot is allocated for each XROO call even if the file is already open. The file slot is allocated beginning with slot 1 to 32.

Possible Errors:

- 50 = Invalid file name
- 53 = File not defined
- 61 = File already open
- 68 = Not PDOS disk
- 69 = Not enough file slots
- Disk errors

4.3.83 XROP - OPEN RANDOM

```
Mnemonic:   XROP
   Value:   $A0E8
  Module:   MPDOSF
  Format:   XROP
              <status error return>

Registers: In   (A1) = File name
           Out  D0.W = File attribute
                D1.W = File ID

      Note: Uses multiple directory file search.
```

```
        LEA.L   FN(PC),A1 ;GET FILE NAME
        XROP              ;OPEN RANDOM FILE
          BNE.S ERROR     ;ERROR
        MOVE.W  D0,D5     ;SAVE TYPE
        SWAP    D5
        MOVE.W  D1,D5     ;SAVE FILE ID
        ....

FN      DC.B    'FILENAME:EXT',0
        EVEN
```

The OPEN RANDOM FILE primitive opens a file for random access by assigning the file to an area of system memory called a file slot, and returning a file ID and file attribute to the calling program. Thereafter, the file is referenced by the file ID and not by the file name.

The file ID (returned in register D1) is a 2-byte number. The left byte is the disk number and the right byte is the channel buffer index. The file attribute is returned in register D0.

```
D0.W = (ABOS BETU xxxx xCWD)
D1.W = (Disk #) x 256 + (File slot index)
```

The END-OF-FILE marker on a random file is changed only when the file has been extended. All data transfers are buffered through a channel buffer and data movement to and from the disk is by full sectors.

The file slot is allocated beginning with slot 32 to slot 1. If the file is already open, then the file slot is shared.

Possible Errors:

50 = Invalid file name
53 = File not defined
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors

4.3.84 XRPS - READ PORT STATUS

```
  Mnemonic:     XRPS
     Value:     $A094
    Module:     MPDOSK2
    Format:     XRPS
                  <status error return>

Registers: In   D0.W = Port number
          Out   D1.L = ACI$.W / portflag.B / Status.B

Note: If D0.W=0, then the current port (PRT$(A6)) is
      used.
```

```
        MOVEQ.L #0,D0    ;LOOK AT CURRENT PORT
        XRPS
          BNE.S ERROR
        BTST.B  #0,D1    ;^S^Q?
          BNE.S CSCQ     ;Y
```

The READ PORT STATUS primitive reads the current status of the port specified by data register D0.W. The high order word of data register D1.L is returned zero if no procedure file is open. Otherwise, it is returned with ACI$.

```
portflag. = fwpi 8dcs
         \\\\ \\\\_ 0 = ^S^Q enable
          \\\\ \\\_ 1 = Control character disable
           \\\\ \\_ 2 = DTR enable
            \\\\ \_ 3 = 8-bit character enable
             \\\\__ 4 = Receiver interrupt enable
              \\\__ 5 = Even parity enable
               \\__ 6 = High water set
                \__ 7 = ^S set
```

The low order word is returned with the port flag bits and the status as returned for the port UART routine. The flag bits indicate if eight bit I/O is occurring, if DTR or ^S ^Q protocol is in effect, and other flags.

See also:

4.3.3 XBCP - BAUD CONSOLE PORT
4.3.98 XSPF - SET PORT FLAG

Possible Errors:

66 = Invalid port or baud rate

4.3.85 XRSE - READ SECTOR

```
      Mnemonic:    XRSE
         Value:    $A0C2
        Module:    MPDOSF
        Format:    XRSE
                     &lt;status error return&gt;

  Registers: In    D0.B = Disk number
                   D1.W = Sector number
                   (A2) = Buffer pointer
```

The READ SECTOR primitive calls a system-defined, hardware-dependent program which reads 256 bytes of data into a memory buffer pointed to by address register A2. The disk is selected by data register D0. Register D1 specifies the logical sector number to be read.

See also:

CHAPTER 8 BIOS
4.3.46 XISE - INITIALIZE SECTOR
4.3.88 XRSZ - READ SECTOR ZERO
4.3.118 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

```
        CLR.W    D0              ;SELECT DISK #0
        MOVEQ.L  #2,D1           ;SELECT SECTOR 2
        LEA.L    BUFF(PC),A2     ;POINT TO BUFFER
        XRSE                     ;READ INTO BUFFER
          BNE.S  XERR            ;ERROR
        ....
XERR    XERR                     ;DISK ERROR
BUFFER  DS.B     256             ;BUFFER
```

4.3.86 XRSR - READ STATUS REGISTER

Mnemonic:	XRSR
Value:	$A042
Module:	MPDOSK1
Format:	XRSR

```
XRSR              ;READ SR
ANDI.W  #$0700,D0
```

Registers: Out D0.W = 68000 status register

The READ STATUS REGISTER primitive allows you to read the 68000 status register. Of course, this is equivalent to the 'MOVE.W SR,Dx' instruction on the 68000. However, this instruction is privileged on the 68010 and 68020. Hence, it is advisable to use the XRSR primitive to read the status register to make software upward compatible.

Possible Errors: None

4.3.87 XRST - RESET DISK

```
 Mnemonic:     XRST
    Value:     $A0B4
   Module:     MPDOSF
   Format:     XRST

Registers: In  D1.W = -1.... Reset by task
                      >=0... Reset by disk
```

The RESET DISK primitive closes all open files either by task or disk number. The primitive also clears the assigned input file ID. If register D1 equals -1, then all files associated with the current task are closed. Otherwise, register D1 specifies a disk and all files opened on that disk are closed.

XRST has no error return and as such, closes all files even though errors occur in the close process. This is necessary to allow for recovery from previous errors.

See also:

4.3.13 XCFA - CLOSE FILE W/ATTRIBUTE
4.3.16 XCLF - CLOSE FILE

Possible Errors: None

```
DONE    MOVEQ.L #-1,D1   ;CLOSE ALL TASK FILES
        XRST
        ....

        MOVE.W  D5,D1    ;PREPARE TO REMOVE DISK
        XRST             ;CLOSE ALL FILES
        ....             ;REMOVE DISK
```

4.3.88 XRSZ - READ SECTOR ZERO

```
Mnemonic:      XRSZ
   Value:      $A0C4
  Module:      MPDOSF
  Format:      XRSZ
                 <status error return>

Registers: In   D0.B = Disk number
           Out  D1.L = 0
                (A2) = User buffer pointer (A6)
```

```
MOVEQ.L #1,D0    ;SELECT DRIVE 1
XRSZ             ;READ HEADER
  BNE.S ERROR
XPBC             ;PRINT DISK NAME
```

The READ SECTOR ZERO primitive is a system-defined, hardware-dependent program which reads 256 bytes of data into the user memory buffer (usually pointed to by address register A6). The disk is selected by data register D0.W. Register D1.L is cleared and logical sector zero is read.

See also:

CHAPTER 8 BIOS
4.3.46 XISE - INITIALIZE SECTOR
4.3.85 XRSE - READ SECTOR
4.3.118 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

4.3.89 XRTE - RETURN FROM INTERRUPT

```
     Mnemonic:    XRTE
        Value:    $A044
       Module:    MPDOSK1
       Format:    XRTE

Registers: In     SSP = Status register.W
                        Program counter.L
```

The RETURN FROM INTERRUPT primitive is used to return from an interrupt process routine with a context switch. This allows an immediate rescheduling of the highest priority ready task which may be suspended pending the occurrence of an event set by the interrupt routine.

If the interrupted system is locked when the XRTE primitive is executed, then the reschedule flag (RFLG.(A5)) is cleared and a return from exception instruction (RTE) is executed. When the system clears the task lock, RFLG. is tested and set (TAS) and a rescheduling occurs at that time.

Possible Errors: None

```
....              ;PROCESS INTERRUPT
MOVEQ.L #66,D1
XSEV              ;SET EVENT 66
XRTE              ;RETURN FROM INTERRUPT
```

4.3.90 XRTM - READ TIME

Mnemonic:	XRTM
Value:	$A05E
Module:	MPDOSK3
Format:	XRTM

```
Registers: Out      (A1) = 'HR:MN:SC'<null>
                10(A1).W = Tics/second (B.TPS)
                12(A1).L = Tics (TICS.)
```

The READ TIME primitive returns the current time as a nine-character string. The format is 'HR:MN:SC' followed by a null. Address register A1 points to the string in the monitor work buffer.

See also:

4.3.32 XFTD - FIX TIME & DATE
4.3.57 XPAD - PACK ASCII DATE
4.3.77 XRDT - READ DATE
4.3.107 XUAD - UNPACK ASCII DATE
4.3.108 XUDT - UNPACK DATE
4.3.112 XUTM - UNPACK TIME

Possible Errors: None

```
GETD    XPMC    MES1    ;OUTPUT PROMPT
        XRTM            ;GET TIME
        XPLC            ;OUTPUT TO SCREEN
        ....

MES1    DC.B    'TIME=',0
```

4.3.91 XRTP - READ TIME PARAMETERS

Mnemonic: XRTP
Value: $A034
Module: MPDOSK1
Format: XRTP

Registers: Out D0.L = TICS.
D1.L = MONTH/DAY/YEAR/0
D2.L = HOURS/MINUTES/SECONDS/0
D3.L = B.TPS

The READ TIME PARAMETERS primitive returns the current time parameters. Data register D0 returns with the current tic count (TICS.(A5)). Register D1.L returns with the current date and register D2.L the current time. Both are three bytes that are left-justified. Finally, data register D3.L returns with the number of clock tics per second.

See also:

4.3.32 XFTD - FIX TIME & DATE
4.3.57 XPAD - PACK ASCII DATE
4.3.77 XRDT - READ DATE
4.3.90 XRTM - READ TIME
4.3.107 XUAD - UNPACK ASCII DATE
4.3.108 XUDT - UNPACK DATE
4.3.112 XUTM - UNPACK TIME

Possible Errors: None

4.3.92 XRTS - READ TASK STATUS

```
Mnemonic:     XRTS
   Value:     $A012
  Module:     MPDOSK1
  Format:     XRTS
                <status return>

Registers: In   D0.W = Task number
           Out  D1.L = 0 - Not executing
                     = +N - Time slice
                     = -N - (Event #1/Event #2)
                A0.L = TLST entry (IF -D0: A0=TLST.)
                  SR = Status of D1.L

Note: If D0.W=-1, then the current task number is
      returned in D1.L.
```

```
WAIT    MOVEQ.L #2,D0    ;WAIT TO TASK 0
        XRST             ; TO DIE
          BNE.S WAIT     ;STILL GOING
        ...              ;DONE
```

The READ TASK STATUS primitive returns in register D1 and the status register returns the time parameter of the task specified by register D0. The time reflects the execution mode of the task. If D1 returns zero, then the task is not in the task list. If D1 returns a value greater than zero, then the task is in the run state (executing). If D1 returns a negative value, then the task is suspended pending event -(D1).

The task number is returned from the CREATE TASK BLOCK (XCTB) primitive. It can also be obtained by setting data register D0 equal to a minus one. In this case, register D1.L is returned with the current task number.

See also:

4.3.100 XSTP - SET/READ TASK PRIORITY

Possible Errors: None

4.3.93 XRWF - REWIND FILE

Mnemonic: XRWF
Value: $A0EA
Module: MPDOSF
Format: XRWF
<status error return>

```
REWIND  MOVE.W  D5,D1     ;GET FILE ID
        XRWF              ;REWIND FILE
          BNE.S ERROR     ;PROBLEM
        ....
```

Registers: In D1.W = File ID

The REWIND FILE primitive positions the file specified by the file ID in register D1, to byte position zero.

See also:

4.3.69 XPSF - POSITION FILE
4.3.79 XRFP - READ FILE POSITION

Possible Errors:

52 = File not open
59 = Invalid slot #
70 = Position error
Disk errors

4.3.94 XSEF - SET EVENT FLAG W/SWAP

```
Mnemonic:      XSEF
   Value:      $A018
  Module:      MPDOSK1
  Format:      XSEF
                 <status return>

Registers: In   D1.B = Event (+=Set, -=Reset)
           Out    SR = NE....Set
                       EQ....Reset
```

```
MOVEQ.L #30,D1  ;SET EVENT 30
XSEF            ;SET EVENT
....

MOVEQ.L #-35,D1 ;RESET EVENT 35
XSEF            ;SET EVENT
....
```

Note: An XSWP is automatically executed after the event is set or reset. Event 128 is local to each task.

If D1.B is positive, then the event is set.
If D1.B is negative, then the event is reset.

The SET EVENT FLAG WITH SWAP primitive sets or resets an event flag bit. The event number is specified in data register D1.B and is modulo 128. If the content of register D1.B is positive, then the event bit is set to 1. Otherwise, the bit is reset to 0. Event 128 can only be set. (It is cleared by the task scheduler.)

4 types of event flags:

```
  1-63 = Software
 64-80 = Software resetting
81-127 = System
   128 = Local to task
```

The status of the event bit prior to changing the event is returned in the status register. If the event was 0, then the 'EQ' status is returned. Also, an immediate context switch occurs thus scheduling any higher priority task pending on that event.

Events are summarized as follows:

```
   1-63 = Software events
  64-80 = Software resetting events
  81-95 = Output port events
 96-111 = Input port events
    112 = 1/5 second event
    113 = 1 second event
    114 = 10 second event
    115 = 20 second event
    116 = TTA active
    117 = LPT active
    118 =
    119 =
    120 = Level 2 lock
    121 = Level 3 lock
    122 = Batch event
    123 = Spooler event
    124 =
    125 =
    126 = Error message disable
    127 = System utility
    128 = Local
```

See also:

4.3.20 XDEV - DELAY SET/RESET EVENT
4.3.95 XSEV - SET EVENT FLAG
4.3.101 XSUI - SUSPEND UNTIL INTERRUPT
4.3.106 XTEF - TEST EVENT FLAG

Possible Errors: None

4.3.95 XSEV - SET EVENT FLAG

```
Mnemonic:     XSEV
   Value:     $A046
  Module:     MPDOSK1
  Format:     XSEV
                <status return>

Registers: In  D1.B = Event (+=Set, -=Reset)
           Out   SR = NE....Set
                      EQ....Reset
```

```
MOVEQ.L #30,D1  ;SET EVENT 30
XSEV            ;SET EVENT
....

MOVEQ.L #-35,D1 ;RESET EVENT 35
XSEV            ;SET EVENT
....
```

Note: Event 128 is local to each task.

If D1.B is positive, then the event is set.
If D1.B is negative, then the event is reset.

The SET EVENT FLAG primitive sets or resets an event flag bit. The event number is specified in data register D1.B and is modulo 128. If the content of register D1.B is positive, then the event bit is set to 1. Otherwise, the bit is reset to 0. Event 128 can only be set. (It is cleared by the task scheduler.)

4 types of event flags:

```
  1-63 = Software
 64-80 = Software resetting
81-127 = System
   128 = Local to task
```

The status of the event bit prior to changing the event is returned in the status register. If the event was 0, then the 'EQ' status is returned. A context switch DOES NOT occur with this call making it useful for interrupt routines outside the PDOS system.

Events are summarized as follows:

```
  1-63 = Software events
 64-80 = Software resetting events
 81-95 = Output port events
96-111 = Input port events
   112 = 1/5 second event
   113 = 1 second event
   114 = 10 second event
   115 = 20 second event
   116 = TTA active
   117 = LPT active
   118 =
   119 =
   120 = Level 2 lock
   121 = Level 3 lock
   122 = Batch event
   123 = Spooler event
   124 =
   125 =
   126 = Error message disable
   127 = System utility
   128 = Local
```

See also:

4.3.20 XDEV - DELAY SET/RESET EVENT
4.3.95 XSEV - SET EVENT FLAG
4.3.101 XSUI - SUSPEND UNTIL INTERRUPT
4.3.106 XTEF - TEST EVENT FLAG

Possible Errors: None

4.3.96 XSMP - SEND MESSAGE POINTER

```
     Mnemonic:    XSMP
        Value:    $A002
       Module:    MPDOSK1
       Format:    XSMP
                    <status return>

Registers: In     D0.B = Message slot number (0..15)
                  (A1) = Message
           Out      SR = EQ....Message sent (Event[64+slot #]=1)
                         NE....No message sent
```

The SEND MESSAGE POINTER primitive sends a 32-bit message to the message slot specified by data register D0.B. Address register A1 contains the message.

If there is still a message pending, then the primitive immediately returns with status set 'Not Equal' and D0.L equal to 83. Otherwise, the message is taken by PDOS event (64 + message slot number) is set to one indicating a message is ready, and status is returned 'Equal'.

The primitive XSMP is only valid for message slots 0 through 15. (This is because of current event limitations.)

See also:

- 4.3.42 XGMP - GET MESSAGE POINTER
- 4.3.44 XGTM - GET TASK MESSAGE
- 4.3.48 XKTM - KILL TASK MESSAGE
- 4.3.99 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message buffer pending

4.3.97 XSOP - OPEN SEQUENTIAL FILE

```
Mnemonic:      XSOP
   Value:      $A0EC
  Module:      MPDOSF
  Format:      XSOP
                 <status error return>

Registers: In  (A1) = File name
           Out D0.W = File attribute
               D1.W = File ID
```

Note: Uses multiple directory file search.

```
        LEA.L   FN(PC),A1 ;GET FILE NAME
        XSOP              ;OPEN SEQUENTIAL FILE
          BNE.S ERROR     ;ERROR
        MOVE.W  D0,D5     ;SAVE TYPE
        SWAP    D5
        MOVE.W  D1,D5     ;SAVE FILE ID
        ....

FN      DC.B    'FILENAME:EXT',0
        EVEN
```

The OPEN SEQUENTIAL FILE primitive opens a file for sequential access by assigning the file to an area of system memory called a file slot and returning a file ID and file type to the calling program. Thereafter, the file is referenced by the file ID and not by the file name.

The file ID (returned in register D1) is a 2-byte number. The left byte is the disk number and the right byte is the file slot index. The file attribute is returned in D0.

D0.W = (ABOS BETD xxxx xCWD)
D1.W = (Disk #) x 256 + (File slot index)

The END-OF-FILE marker on a sequential file is changed whenever data is written to the file. All data transfers are buffered through a channel buffer; data movement to and from the disk is by full sectors.

The file slots are allocated beginning with slot 32 down to slot 1.

Possible Errors:

50 = Invalid file name
53 = File not defined
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors

4.3.98 XSPF - SET PORT FLAG

```
Mnemonic:     XSPF
   Value:     $A09A
  Module:     MPDOSK2
  Format:     XSPF
                &lt;status error return&gt;
```

```
MOVEQ.L #0,D0    ;SELECT CURRENT
MOVEQ.L #1,D1    ;^S^Q
XSPF
```

```
Registers: In   D0.W = Port number
                D1.B = Port flag (fwpi8dcs)
          Out   D1.B = Old port flag
```

Note: If D0.W=0, then the current port (PRT$(A6)) is used.

The SET PORT FLAG primitive stores the port flag passed in data register D1.B in the port flag register as specified by register D0.W.

If flag bits 'p', 'i', or '8' change, the BIOS baud port routine is called.

See also:

4.3.3 XBCP - BAUD CONSOLE PORT
4.3.84 XRPS - READ PORT STATUS

Possible Errors:

66 = Invalid port or baud rate

4.3.99 XSTM - SEND TASK MESSAGE

```
    Mnemonic:     XSTM
       Value:     $A020
      Module:     MPDOSK1
      Format:     XSTM
                    &lt;status error return&gt;

Registers: In     D0.B = TASK NUMBER
                  (A1) = MESSAGE
```

The SEND TASK MESSAGE primitive places a 64-character message into a PDOS system message buffer. The message is data-independent and is pointed to by address register A1.

Data register D0 specifies the destination of the message. If register D0 is negative, and there is no input port (phantom port), then the message is sent to the parent task. If there is a port, then the message is sent to itself and will appear at the next command line. Otherwise, register D0 specifies the destination task.

The ability to direct a message to a parent task is very useful in background tasking. An assembler need not know from which task it was spawned and can merely direct any diagnostics to the parent task.

If the destination task number equals -1, the task message is moved to the monitor input buffer and parsed as a command line. This feature is used by the CREATE TASK BLOCK primitive to spawn a new task.

See also:

- 4.3.42 XGMP - GET MESSAGE POINTER
- 4.3.44 XGTM - GET TASK MESSAGE
- 4.3.48 XKTM - KILL TASK MESSAGE
- 4.3.96 XSMP - SEND MESSAGE POINTER
- 4.3.99 XSTM - SEND TASK MESSAGE

Possible Errors:

78 = Message buffer full

```
TERR    LEA.L   ERRM(PC),A1      ;RETURN MESSAGE
        ST.B    D0               ;SEND TO PARENT
        XSTM                     ;SEND, ERROR?
          BNE.S ERROR            ;Y
        XEXT                     ;N, QUIT
```

D0 = -1 sends message to parent task

4.3.100 XSTP - SET/READ TASK PRIORITY

```
Mnemonic:      XSTP
   Value:      $A03C
  Module:      MPDOSK1
  Format:      XSTP
                 <status error return>

Registers: In  D0.B = Task #
               D1.W = Task time/Task priority
          Out  D1.B = Task priority (If D1.B was 0)

Note: If D0.B=-1, then select current task.
      If D1.B=0, then read task priority into D1.B.
```

The SET/READ TASK PRIORITY primitive either sets or reads the task priority selected by data register D0.B. If D1.B is nonzero, then the priority is set. Otherwise, it is read and returned in D1.B. If the upper byte of D1.W is nonzero, then the corresponding task time slice is also set.

See also:

4.3.92 XRTS - READ TASK STATUS

Possible Errors:

74 = No such task

```
MOVEQ.L #-1,D0  ;CURRENT TASK
MOVEQ.L #0,D1   ;SET TO READ
XSTP            ;READ TASK PRIORITY
  BNE.S ERROR
MOVE.B  D1,SV(A2)
....
```

```
MOVEQ.L #-1,D0  ;SELECT CURRENT
MOVEQ.L #100,D1 ;SET TO WRITE
XSTP            ;SET TASK PRIORITY
  BNE.S ERROR
```

4.3.101 XSUI - SUSPEND UNTIL INTERRUPT

```
Mnemonic:      XSUI
   Value:      $A01C
  Module:      MPDOSK1
  Format:      XSUI

Registers: In  D1.W = EV1/EV2
           Out D0.L = Event
```

The SUSPEND UNTIL INTERRUPT primitive suspends the user task until one of the events specified in data register D1 occurs. A task can suspend until an event sets (positive event) or until it resets (negative event).

A task can suspend pending two different events. This is useful when combined with timeout counters to prevent system lockups. Data register D0.L is returned with the event which caused the task to be scheduled.

A suspended task does not receive any CPU cycles until one of the event conditions is met. When the event bit is set (or reset), the task begins executing at the next instruction after the XSUI call. The task is scheduled during the normal swapping functions of PDOS according to its priority. Register D0.L is used to determined which event scheduled the task.

A suspended task is indicated in the LIST TASK (LT) command under the 'Event' parameter. Multiple events are separated by a slash.

Events 64 through 128 toggle when they cause a task to move from the suspended state to the ready state. All others must be reset by the event routine.

If a locked task attempts to suspend itself, the call polls the events until a successful return condition is met.

See also:

4.3.20 XDEV - DELAY SET/RESET EVENT
4.3.94 XSEF - SET EVENT FLAG W/SWAP
4.3.95 XSEV - SET EVENT FLAG
4.3.106 XTEF - TEST EVENT FLAG

Possible Errors: None

```
GETC    XGCC                    ;CHARACTER?
          BNE.S GETC2           ;Y
        MOVEQ.L #100,D0         ;N, GET DELAY
        MOVEQ.L #128,D1         ;USER LOCAL EVENT
        XDEV                    ;DELAY 128 1 SECOND
          BNE.S GETC            ;FULL
        LSL.W   #8,D1           ;GET 128/(PORT+96)
        MOVE.B  #96,D1
        ADD.B   PRT$(A6),D1
        XSUI                    ;SUSPEND
        CMP.B   D0,D1           ;CHARACTER EVENT?
          BEQ.S GETC            ;Y
```

4.3.102 XSUP - ENTER SUPERVISOR MODE

Mnemonic:	XSUP
Value:	$A02C
Module:	MPDOSK1
Format:	XSUP
Registers:	None

The ENTER SUPERVISOR MODE primitive moves your current task from user mode to supervisor mode. Care should be taken not to crash the system since you would then be executing off the supervisor stack!

This primitive enables programs to access I/O addresses and use privileged instructions.

You exit to user mode by executing a 'ANDI.W #$DFFF,SR' instruction or the XUSP primitive.

See also:

4.3.54 XLSR - LOAD STATUS REGISTER
4.3.112 XUSP - RETURN TO USER MODE

Possible Errors: None

```
P1      EQU $FFFFCE01   ;I/O PORT
*
OUT     XSUP            ;ENTER SUPERVISOR
        MOVE.B  D0,P1   ;OUTPUT
        ANDI.W  #$DFFF,SR ;MOVE TO USER
        RTS             ;RETURN
```

4.3.103 XSWP - SWAP TO NEXT TASK

Mnemonic:	XSWP
Value:	$A000
Module:	MPDOSK1
Format:	XSWP
Registers:	None

The SWAP TO NEXT TASK primitive relinquishes control to the PDOS task scheduler. The next ready task with the highest priority begins executing. (This may be to the same task if there is only one task or the task is the highest priority ready task.)

Possible Errors: None

```
LOOP     TST.B    TMEM      ;CONDITION MET?
           BEQ.S LOOP02     ;Y
         XSWP               ;N, SWAP WHILE WAITING
         BRA.S    LOOP
*
LOOP02   ....
```

4.3.104 XSZF - GET DISK SIZE

```
     Mnemonic:    XSZF
        Value:    $A0B6
       Module:    MPDOSF
       Format:    XSZF
                    &lt;status error return&gt;

Registers: In     D0.B = Disk number
           Out    D5.L = Directory size/# of files
                  D6.L = Allotted/Used
                  D7.L = Largest/Free
```

The GET DISK SIZE primitive returns disk size parameters in data registers D5 through D7. Data register D5 returns the number of currently defined files in the low word along with the maximum number of files available in the directory in the high word.

The low order 16 bits of data register D6 (0-15) returns the total number of sectors used by all files. The high order 16 bits of D6 (16-31) returns the number of sectors allocated for file storage.

The low order 16 bits of data register D7 (0-15) is calculated from the disk sector bit map and reflects the number of sectors available for file allocation. The high order 16 bits of D7 (16-31) is returned with the size of the largest block of contiguous sectors. This is useful in defining large files.

Possible Errors:

68 = Not PDOS disk
Disk errors

```
        CLR.L   D0        ;SELECT DISK #0
        XSZF              ;GET DISK SIZE
          BNE.S ERROR     ;ERROR
        CLR.L   D1
        MOVE.W  D7,D1
        XCBM    SPM1      ;OUTPUT FREE
        XPLC              ;PRINT
        SWAP    D7
        MOVE.W  D7,D1
        XCBM    SPM2      ;OUTPUT LARGEST
        XPLC              ;  CONTIGUOUS BLOCK
        XTAB    20        ;TAB TO COLUMN 20
        MOVE.W  D6,D1
        XCBM    SPM3      ;OUTPUT USED
        XPLC              ;PRINT
        SWAP    D6
        MOVE.W  D6,D1
        XCBM    SPM4      ;OUTPUT ALLOCATED
        XPLC              ;PRINT
        XEXT
*
SPM1    DC.B    $0A,$0D,'FREE:',0
SPM2    DC.B    ',',0
SPM3    DC.B    'USED:',0
SPM4    DC.B    '/',0
        EVEN
```

4.3.105 XTAB - TAB TO COLUMN

Mnemonic:	XTAB
Value:	$A090
Module:	MPDOSK2
Format:	XTAB <column>
Registers:	None

The TAB TO COLUMN primitive positions the cursor to the column specified by the number following the call. Spaces are output until the column counter is greater than or equal to the parameter.

The first print column is zero. At least one space character will always be output.

Possible Errors: None

```
XPMC    MES1    ;OUTPUT HEADER
XTAB    30      ;MOVE TO COLUMN 30
....
```

4.3.106 XTEF - TEST EVENT FLAG

```
Mnemonic:     XTEF
   Value:     $A01A
  Module:     MPDOSK1
  Format:     XTEF
                <status return>

Registers: In   D1.B = Event number (+=0-127, -=128)
           Out   SR = NE....Event set (1)
                      EQ....Event clear (0)
```

```
MOVEQ.L #30,D1   ;EVENT 30
XTEF             ;TEST EVENT FLAG
  BNE.S EVENT    ;EVENT = .TRUE.
....             ;EVENT = .FALSE.
```

The TEST EVENT FLAG primitive sets the 68000 status word EQUAL or NOT-EQUAL depending upon the zero or nonzero state of the specified event flag. The flag is not altered by this primitive.

The event number is specified in data register D1 and is modulo 128. Event 128 is local to each task.

See also:

4.3.20 XDEV - DELAY SET/RESET EVENT
4.3.94 XSEF - SET EVENT FLAG W/SWAP
4.3.95 XSEV - SET EVENT FLAG
4.3.101 XSUI - SUSPEND UNTIL INTERRUPT

Possible Errors: None

4.3.107 XUAD - UNPACK ASCII DATE

```
     Mnemonic:   XUAD
        Value:   $A036
       Module:   MPDOSK3
       Format:   XUAD

Registers: In    D1.W = (Year*16+Month)*32+Day
                        (YYYY YYYM MMMD DDDD)
           Out   (A1) = 'DY-MON-YR'<null>
                        (Outputs ??? for invalid months)
```

The UNPACK ASCII DATE primitive returns a pointer in address register A1 to an ASCII date string. Data register D1.W contains the binary date [(Year*16+Month)*32+Day]. The format of the string is more exact than simple numbers separated by slashed.

Note: XUAD does not check for a valid date and hence, funny looking strings could result. Invalid months are replaced by '???.'

See also:

4.3.32 XFTD - FIX TIME & DATE
4.3.57 XPAD - PACK ASCII DATE
4.3.77 XRDT - READ DATE
4.3.90 XRTM - READ TIME
4.3.108 XUDT - UNPACK DATE
4.3.112 XUTM - UNPACK TIME

Possible Errors: None

4.3.108 XUDT - UNPACK DATE

```
Mnemonic:   XUDT
   Value:   $A060
  Module:   MPDOSK3
  Format:   XUDT

Registers: In   D1.W = (Year * 16 + Month) * 32 + Day
           Out  (A1) = 'MN/DY/YR'<null>
```

```
XFTD        ;FIX TIME & DATE
XUDT        ;UNPACK DATE
XPLC        ;PRINT 'MN/DY/YR'
....
```

The UNPACK DATE primitive converts a one-word encoded date into an eight-character string terminated by a null (nine characters total). Data register D1 contains the encoded date and returns with a pointer to the formatted string in address register A1. The output of the FIX TIME & DATE (XFTD) primitive is valid input to this primitive.

See also:

4.3.32 XFTD - FIX TIME & DATE
4.3.57 XPAD - PACK ASCII DATE
4.3.77 XRDT - READ DATE
4.3.90 XRTM - READ TIME
4.3.107 XUAD - UNPACK ASCII DATE
4.3.112 XUTM - UNPACK TIME

Possible Errors: None

4.3.109 XULF - UNLOCK FILE

```
Mnemonic:   XULF
   Value:   $A0EE
  Module:   MPDOSF
  Format:   XULF
              <status error return>

Registers: In   D1.W = File ID
```

```
MOVE.W  D5,D1   ;GET FILE ID
XULF            ;UNLOCK FILE
  BNE.S ERROR
....
```

The UNLOCK FILE primitive unlocks a locked file for access by any other task. The file is specified by the file ID in data register D1.

See also:

4.3.52 XLKF - LOCK FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
Disk errors

4.3.110 XULT - UNLOCK TASK

Mnemonic:	XULT
Value:	$A016
Module:	MPDOSK1
Format:	XULT
Registers:	None

The UNLOCK TASK primitive unlocks the current task by clearing the swap lock variable in system RAM. This allows other tasks to be scheduled and receive CPU time.

See also:

4.3.53 XLKT - LOCK TASK

Possible Errors: None

```
        XLKT            ;LOCK TASK WHILE WAITING
*
LOOP    TST.B   LMEM    ;CONDITION MET?
          BNE.S LOOP    ;N, WAIT
        CLR.B   OMEM    ;Y, RESET
        XULT            ;UNLOCK TASK NOW
```

4.3.111 XUSP - RETURN TO USER MODE

Mnemonic: XUSP
Value: $A008
Module: MPDOSK1
Format: XUSP

Registers: None

The RETURN TO USER MODE primitive moves your current task from supervisor mode to user mode. Executing an 'ANDI.W #$DFFF,SR'' instruction also returns you to user mode, but must be executed in supervisor mode. The XUSP primitive can be executed in either mode.

See also:

4.3.54 XLSR - LOAD STATUS REGISTER
4.3.103 XSUP - ENTER SUPERVISOR MODE

Possible errors: None

```
P1      EQU $FFFFCE01     ;1/0 PORT
*
OUT     XSUP              ;ENTER SUPERVISOR
        MOVE.B D0,P1      ;OUTPUT
        XUSP              ;RETURN TO USER
        RTS               ;RETURN
```

4.3.112 XUTM - UNPACK TIME

```
    Mnemonic:    XUTM
       Value:    $A062
      Module:    MPDOSK3
      Format:    XUTM

Registers: In    D1.W = HOUR*256+MINUTE
                        (HHHH HHHH MMMM MMMM)
          Out    (A1) = HR:MN<null>
```

```
XFTD            ;GET SYSTEM TIME
MOVE D0,D1
XUTM            ;CONVERT TO STRING
XPLC            ;PRINT TIME
....
```

The UNPACK TIME primitive converts a one word encoded date into a five character string terminated by a null (six characters total). Data register D1 contains the encoded time and returns a pointer to the formatted string in address register A1. The output of the FIX TIME & DATE (XFTD) primitive is valid input to this primitive.

See also:

- 4.3.32 XFTD - FIX TIME & DATE
- 4.3.57 XPAD - PACK ASCII DATE
- 4.3.77 XRDT - READ DATE
- 4.3.90 XRTM - READ TIME
- 4.3.107 XUAD - UNPACK ASCII DATE
- 4.3.108 XUDT - UNPACK DATE

Possible Errors: None

4.3.113 XWBF - WRITE BYTES TO FILE

```
Mnemonic:    XWBF
   Value:    $A0F0
  Module:    MPDOSF
  Format:    XWBF
               <status error return>

Registers: In  D0.L = Byte count - must be positive
               D1.W = File ID
               (A2) = Buffer address
```

```
        MOVE.L  #252,D0      ;WRITE FULL SECTOR
        MOVE.W  D5,D1        ;GET ID
        LEA.L   BF(PC),A2    ;GET BUFFER ADDRESS
        XWBF                 ;WRITE TO FILE
          BNE.S ERROR
        ....

BF      DS.B    256          ;SECTOR BUFFER
```

The WRITE BYTES TO FILE primitive writes from a memory buffer, pointed to by address register A2, to a disk file specified by the file ID in register D1. Register D0 specifies the number of bytes to be written. If the channel buffer has been rolled to disk, the least-used buffer is freed and the buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

The write is independent of the data content. The buffer pointer in register A2 may be on any byte boundary. The write operation is not terminated with a null character.

A byte count of zero in register D0 results in no data being written to the file.

D0 = 0 Write no data

If it is necessary for the file to be extended, PDOS first uses sectors already linked to the file. If a null or end link is found, a new sector obtained from the disk sector bit map is linked to the end of the file. If this makes the file non-contiguous, it is retyped as a non-contiguous file.

Extended file

Contiguous changes to non-contiguous

See also:

4.3.71 XRBF - READ BYTES FROM FILE
4.3.80 XRLF - READ LINE FROM FILE
4.3.117 XWLF - WRITE LINE TO FILE

Possible Errors:

52 = File not open
58 = File delete or write protected
59 = Invalid slot #
60 = File space full
Disk errors

4.3.114 XWDT - WRITE DATE

Mnemonic:	XWDT
Value:	$A064
Module:	MPDOSK3
Format:	XWDT
Registers: In	D0.B = Month (1-12)
	D1.B = Day (1-31)
	D2.B = Year (0-99)

```
MOVEQ.L #12,D0   ;SET DATE TO 12/25/80
MOVEQ.L #25,D1
MOVEQ.L #83,D2
XWDT             ;SET DATE
....
```

The WRITE DATE primitive sets the system date counters. Register D0 specifies the month and ranges from 1 to 12. Register D1 specifies the day of month and ranges from 1 to 31. Register D2 is the last 2 digits of the year.

No check is made for a valid date.

Possible Errors: None

4.3.114 XWFA - WRITE FILE ATTRIBUTES

```
Mnemonic:   XWFA
   Value:   $A0F2
  Module:   MPDOSF
  Format:   XWFA
              <status error return>

Registers: In  (A1) = File name
               (A2) = ASCII file attributes

     Note: (A2)=0 clears all attributes.
```

The WRITE FILE ATTRIBUTES primitive sets the attributes of the file specified by the file name pointed to by register A1. Register A2 points to an ASCII string containing the new file attributes followed by a null character. The format is:

```
(A2) = {file type}{protection}

     {file type} = AC - Procedure file
                   BN - Binary file
                   OB - 68000 object file
                   SY - 68000 memory image
                   BX - BASIC binary token file
                   EX - BASIC ASCII file
                   TX - Text file
                   DR - System I/O driver

    {protection} = *  - Delete protect
                   ** - Delete and Write protect
```

If register A2 points to a zero byte, then all flags, with the exception of the contiguous flag, are cleared.

See also:

4.3.13 XCFA - CLOSE FILE W/ATTRIBUTE
4.3.78 XRFA - READ FILE ATTRIBUTES
4.3.116 XWFP - WRITE FILE PARAMETERS

Possible Errors:

50 = Invalid file name
53 = File not defined
54 = Invalid file type
Disk errors

```
        LEA.L   FN(PC),A1 ;GET FILE NAME
        LEA.L   PF(PC),A2 ;SET BINARY & PROTECTED
        XWFA              ;SET
          BNE.S ERROR
        ....

FN      DC.B    'DATA:BIN',0
PF      DC.B    'BN**',0
        EVEN
```

4.3.116 XWFP - WRITE FILE PARAMETERS

```
     Mnemonic:    XWFP
        Value:    $A0FC
       Module:    MPDOSF
       Format:    XWFP
                    &lt;status error return&gt;

Registers: In     (A1) = File name
                  D0.L = Sector index of EOF/Bytes in last sector
                  D1.L = Time/Date created
                  D2.L = Time/Date last accessed
                  D3.W = OR'd status (less contiguous bit)
```

The WRITE FILE PARAMETERS primitive updates the end-of-file and date parameters of the file specified by the name pointed to by address register A1 in the disk directory.

See also:

4.3.13 XCFA - CLOSE FILE W/ATTRIBUTE
4.3.78 XRFA - READ FILE ATTRIBUTES
4.3.115 XWFA - WRITE FILE ATTRIBUTES

Possible Errors:

50 = Invalid file name
53 = File not defined
Disk errors

```
        LEA.L    FN(PC),A1    ;GET FILE NAME
        XRFA                  ;READ FILE ATTRIBUTES
          BNE.S ERROR         ;ERROR
        ADDA.W   #20,A2       ;POINT TO
        MOVEM.L (A2),D5-D7    ;SAVE PARAMETERS
        ...

        MOVE.L   D5,D0
        MOVE.L   D6,D1
        MOVE.L   D7,D2
        LEA.L    FN(PC),A1    ;GET FILE NAME
        XWFP                  ;UPDATE FILE PARAMETER
          BNE.S ERROR
        ....

FN      DC.B     'DATA:BIN',0
        EVEN
```

4.3.117 XWLF - WRITE LINE TO FILE

```
Mnemonic:      XWLF
   Value:      $A0F4
  Module:      MPDOSF
  Format:      XWLF
                 <status error return>

Registers: In  D1.W = File ID
               (A2) = Buffer address
```

```
        MOVE.W  D5,D1       ;GET FILE ID
        LEA.L   LB(PC),A2   ;GET LINE
        XWLF                ;WRITE LINE
          BNE.S ERROR       ;ERROR
        ....

LB      DC.B    $0A,$0D,'NO DIAGNOSTICS',0
        EVEN
```

The WRITE LINE TO FILE primitive writes a line delimited by a null character to the disk file specified by the file ID in register D1. Address register A2 points to the string to be written. If the channel buffer has been rolled to disk, the least-used buffer is freed and the buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

The write line primitive is independent of the data content, with the exception that a null character terminates the string. The buffer pointer in register A2 may be on any byte boundary. A single write operation continues until a null character is found.

Null delimiter

If it is necessary for the file to be extended, PDOS first uses sectors already linked to the file. If a null link is found, a new sector obtained from the disk sector bit map is linked to the end of the file. If this makes the file non-contiguous, it is retyped as a non-contiguous file.

Extended file

Contiguous changes to non-contiguous

See also:

- 4.3.71 XRBF - READ BYTES FROM FILE
- 4.3.80 XRLF - READ LINE FROM FILE
- 4.3.113 XWBF - WRITE BYTES TO FILE

Possible Errors:

- 52 = File not open
- 58 = File delete or write protected
- 59 = Invalid slot #
- 60 = File space full
- Disk errors

4.3.118 XWSE - WRITE SECTOR

```
  Mnemonic:       XWSE
     Value:       $A0C6
    Module:       MPDOSF
    Format:       XWSE
                    <status error return>

Registers: In     D0.B = Disk number
                  D1.W = Sector number
                  (A2) = Buffer address
```

The WRITE SECTOR primitive is a system-defined, hardware-dependent program which writes 256 bytes of data from a buffer, pointed to by address register A2, to the logical sector and disk device specified by data registers D1 and D0 respectively.

See also:

CHAPTER 8 BIOS
4.3.46 XISE - INITIALIZE SECTOR
4.3.85 XRSE - READ SECTOR
4.3.88 XRSZ - READ SECTOR ZERO

Possible Errors:

Disk errors

```
        CLR.L   D0            ;WRITE TO DISK #0
        MOVEQ.L #10,D2        ;WRITE TO SECTOR #10
        LEA.L   BUF(PC),A2    ;GET BUFFER ADDRESS
        XWSE                  ;WRITE
          BNE.S ERROR         ;PROBLEM
        ....

BUF     DS.B    256              ;DATA BUFFER
```

4.3.119 XWTM - WRITE TIME

Mnemonic:	XWTM
Value:	$A066
Module:	MPDOSK3
Format:	XWTM
Registers: In	D0.B = Hours (0-23)
	D1.B = Minutes (0-59)
	D2.B = Seconds (0-60)

```
MOVEQ.L #23,D0  ;SET TIME TO 23:59:59
MOVEQ.L #59,D1
MOVEQ.L #59,D2
XWTM            ;SET SYSTEM TIME
```

The WRITE TIME primitive sets the system clock time. Register D0 specifies the hour and ranges from 0 to 23. Register D1 specifies the minutes and register D2, the seconds. The latter two range from 0 to 59.

There is no check made for a valid time.

Possible Errors: None

4.3.120 XZFL - ZERO FILE

```
Mnemonic:   XZFL
   Value:   $A0F6
  Module:   MPDOSF
  Format:   XZFL
              <status error return>

Registers: In   (A1) = File name
```

The ZERO FILE primitive clears a file of any data. If the file is defined, then the end-of-file marker is placed at the beginning of the file. If the file is not defined, it is defined with no data.

See also:

4.3.21 XDFL - DEFINE FILE
4.3.22 XDLF - DELETE FILE

Possible errors:

50 = Invalid file name
61 = File already open
68 = Not PDOS disk
Disk errors

```
         LEA.L FN(PC),A1 ;POINT TO FILE
         XZFL            ;ZERO FILE
           BNE.S ERROR
         ....
FN       DC.B  'FILE:SR',0
         EVEN
```

CHAPTER 5

PDOS SCREEN EDITOR

The MEDIT editor is a screen oriented, memory editor.

5.1 INTRODUCTION

Screen Editor

MEDIT is a screen oriented editor designed for editing files on a CRT terminal. And because MEDIT is a screen editor, what you see is what you get when you print it out. The screen is constantly updated to reflect the current image of the text that you are editing. Since you can always see what your file looks like, you are less likely to become lost or confused than with line- or character-oriented editors.

By just learning a few of the basic commands, you can proficiently edit files. There are also many advanced features that can cut your editing time considerably.

MEDITCON configures MEDIT for function keys. (See section 5.5)

MEDIT may be configured to allow you to use your function and arrow keys. You may configure the editor for your terminal using the MEDITCON utility described in section 5.5.

5.2 GETTING STARTED

Before you can enter the editor, you must make sure that your terminal characteristics are set properly. MEDIT only requires that the terminal be able to clear the screen and position the cursor. The MTERM utility sets the position cursor (PSC$) and clear screen (CSC$) variables in the task control block (TCB). This utility 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 parent task control set. MTERM provides an easy way for a task to change its function characters while the system is running.

```
x>MTERM
68K PDOS Change Terminal Type Utility
Terminals:
    A=ADDS Regent 25
    D=Decscope (VT52)
    H=Hazeltine 1520
    I=Intertube II
    L=Lear Seigler ADM3a
    S=Soroc IQ120
    M=Data Media Excel 12
    V=VT100 / ANSI terminal
    U=User Defined
Type = H
x>
```

If your terminal is not listed, enter a 'U' and enter the hexadecimal representation of the sequences used by your terminal (see the MTERM utility for more details). Terminals which send character sequences longer than four characters require special BIOS support.

Because the editor functions are configurable, this section only refers to the name of the function and the default key sequence in parentheses. An example is QUIT ([ESC][CTRL-V]). You can substitute the key sequence for the function keys you select later.

5.3 THE CLOCK

During editing, the current time is kept in the lower right hand corner of your screen (on the status line). It is only updated when no other processing is occurring and serves as an indicator that the system is alive. When the editor is performing a lengthy process such as writing out a large file, you can watch the clock to see when the process is complete. When it starts ticking again, the editor will accept new commands.

5.4 USING MEDIT

From the PDOS monitor, you can enter the editor by typing 'MEDIT <filename>[CR]'. If you wish to enter new text, just type 'MEDIT[CR]'. To edit an existing file, you may enter the filename directly on the PDOS command line. If you forget to enter the filename, you may also retrieve the file from MEDIT by using the FILE RETRIEVE ([CTRL-G]) command.

>MEDIT {<filename>}

>MEDIT[CR]
FILE RETRIEVE -- [CTRL-G]<filename>[CR]

Following is a description of the MEDIT commands in alphabetical order. These descriptions are intended as a reference guide. To learn to use MEDIT, consult <u>Getting Started With PDOS</u>. The diagram below illustrates the windowing effect of MEDIT.

```
                              Computer memory
                                  ._______.
                       ^T ---> | Text    |          ^D      .____________.
                               |         |      ,-----> |Macro buffer|
                               | ^K v    |     /  ^E    '------------'
                               |    v    |    /
   ._________.          ^W     .--+--------+--. /
  /________/| <-------------- |   Terminal    |      ^U
 | Floppy | | --------------> |    Window     | ---------> .__________.
 |________|/       ^G   ^R ->|               | <--------- |Up buffer|
                              |    24 x 80    |  [ESC]^U   '---------'
                              '--+--------+--'
                                 |        |
                                 |    ^   |
                                 | ^J ^   |
                       ^P ---> |.......|          ^B = Search backwards
                                 |        |          ^F = Search forwards
                       ^Z ---> '-------'
```

Some PDOS users have terminals which allow up to 132 characters on the screen and/or more than 24 lines per screen. MEDIT allows you to select row and column size. The default is 80 columns and 24 lines per screen. To utilize this feature, you can use the two optional row and column arguments:

MEDIT <filename>{<,col><,row>}{<,macro>}

x><u>MEDIT ,132</u>	for 132 columns
x><u>MEDIT FILENAME:SR,,49</u>	for 80 columns and 49 lines

You can also specify the name of a macro on the command line. This will load the macro into the buffer and execute it once.

x><u>MEDIT FILENAME:SR,,,FILE:MAC</u>

BUFFER COMMANDS (CUT AND PASTE)

MEDIT maintains a 4K buffer for moving text from one part of a file to another. Two commands make use of this buffer -- BUFFER FILL ([CTRL-U]) and BUFFER RETRIEVE ([ESC][CTRL-U]). Once text has been copied into the buffer, it may be retrieved any number of times. Copying data into the buffer does not remove it from its original location in the text. To delete text once it has been copied into the buffer, use the DELETE TO POINTER ([ESC][CTRL-\]) command.

BUFFER FILL ([CTRL-U])

This command works in conjunction with the PLACE POINTER ([CTRL-P]) command. To fill the user buffer, place the pointer at the end of the text which is to be placed into the buffer. Position the cursor at the beginning of the text and type [CTRL-U]. MEDIT will respond with either "I got it" which means that the text has been placed successfully into the buffer, or "Overflow" which means that the amount of text was too large to fit into the buffer. If the amount of text was too large, you can either divide the text and place each portion into the buffer in turn (remember that when you place something into the buffer, it overwrites the text that was already there), or you can place the text into a file using the FILE EXCERPT ([CTRL-O]) command.

The text you have placed into the buffer will remain in the buffer until you exit MEDIT or place something else into the buffer.

The pointer is not deleted when you fill the buffer. It should be deleted, however, when not required. The pointer will automatically be deleted when you write the text to a file using the FILE SAVE ([CTRL-W]) command.

Commands used with BUFFER FILL:

- BUFFER RETRIEVE ([ESC][CTRL-U])
- DELETE TO POINTER ([CTRL-\])
- PLACE POINTER ([CTRL-P])

(BUFFER COMMANDS continued)

BUFFER RETRIEVE ([ESC][CTRL-U])

The BUFFER RETRIEVE command inserts the contents of the buffer into the text at the cursor. To use this command, you should have already placed the text you wish to place into the buffer with the BUFFER FILL ([CTRL-U]) command. Then position your cursor at the location in your file where you wish the text to be inserted and type [ESC][CTRL-U]. The text will appear in your file. It does not write over existing text. The cursor remains at the same position that you placed it, not at the end of the inserted text.

Commands used with BUFFER RETRIEVE:

BUFFER FILL ([CTRL-U])

CANCEL ([CTRL-C])

The CANCEL command aborts the current function. Pressing [CTRL-C] does not cause you to exit MEDIT. This command is generally used as cancel in PDOS as well and as such, is best left as [CTRL-C] when configuring MEDIT with the MEDITCON utility.

The CANCEL([CTRL-C]) function is allowed at various points in the editor to let you back out of a function. Generally, anywhere the editor prompts for an argument for a command, a CANCEL([CTRL-C]) will abort the command. CANCEL([CTRL-C]) will also stop a long macro. The PDOS monitor checks specifically for the [CTRL-C] character and clears the type-ahead buffer when it is found, so the [CTRL-C] can break through other commands that might be queued up.

WARNING! CANCEL will abort a FILE SAVE ([CTRL-W]) with the file only partly written. If you then perform a FILE RETRIEVE ([CTRL-G]), you may lose data.

CLEAR EDITOR ([CTRL-N])

The CLEAR EDITOR command clears the buffers and memory so that you have a clean screen to begin editing again. Remember that using this command will erase everything you have on the screen. If you wish to save your current file before you begin with a new file, use the FILE SAVE ([CTRL-W]) command.

After you type [CTRL-N], the editor asks you to verify the command with a 'V'.

The CLEAR EDITOR command does NOT clear the user buffer. As such, you may store text with the BUFFER FILL ([CTRL-U]) command, clear the editor, then using the BUFFER RETRIEVE ([ESC][CTRL-U]) command, retrieve the saved text on a new screen.

Commands used with CLEAR EDITOR:

- FILE SAVE ([CTRL-W])
- BUFFER FILL ([CTRL-U])
- BUFFER RETRIEVE ([ESC][CTRL-U])
- FILE EXCERPT ([CTRL-O])

COMMAND MODE ([ESC][CTRL-C])

This command is not currently implemented; however, if you type [ESC][CTRL-C], MEDIT responds as if a QUIT were typed in. Do not, however, use this command as an exit immediately after saving a file, because the [CTRL-C] will abort the FILE SAVE (if it is still in progress) and you will be left with only part of the text written out.

DELETE COMMANDS

MEDIT provides several ways to remove text from your workspace. Of course, to delete all the text, use the CLEAR EDITOR ([CTRL-N]) command. Otherwise, use one of the following:

DELETE CONTROL CHARACTERS ([ESC][CTRL-N])

DELETE CONTROL CHARACTERS eliminates all control characters (except for [TAB] and [CR]) and clears the eighth bit on all characters. The pointer, if present, will not be affected by this command.

DELETE LEFT [RUB]

DELETE LEFT deletes the character to the left of your cursor, or the character you just typed. You will probably use it the most. If you do not have a key called "rubout" on your terminal, it may be called "delete" or "del".

DELETE RIGHT ([CTRL-UNDERSCORE])

DELETE RIGHT (CTRL-_) deletes the character which is under the cursor, moving the text on the right of the cursor one space to the left.

DELETE LINE ([CTRL-UPARROW])

DELETE LINE (CTRL-^) deletes all the characters to the right of the cursor up to and including the [CR].

DELETE TO EOL ([CTRL-RIGHT SQUARE BRACKET])

DELETE TO EOL (CTRL-]) deletes all the characters to the right of the cursor up to, but not including the next [CR].

DELETE TO POINTER ([CTRL-\])

DELETE TO POINTER deletes all the text from the cursor position either backwards or forwards to the pointer. It is often used in conjunction with other cut and paste commands such as BUFFER FILL ([CTRL-U]). You must position the pointer with the PLACE POINTER ([CTRL-P]) command. MEDIT asks you to verify the command with a 'V.'

FILE INSERT AND EXCERPT COMMANDS

MEDIT allows you to break files up and recombine them into different files by letting you cut sections out of one file and merge them into others. The following describes this pair of commands:

FILE EXCERPT ([CTRL-O])

The FILE EXCERPT command places the text between the pointer and your cursor into a PDOS file. You must first place the pointer with the PLACE POINTER ([CTRL-P]) command at the end of the text you wish to copy to a new file. Then, position your cursor at the beginning of the block of text and type [CTRL-O]. MEDIT will ask you for the name of the file you wish to place the text into. You must name the file a legal PDOS filename. Be sure that you do not name this new file the name of the file you are currently editing or any other file which you wish to keep as it will write over existing files. Terminate the name with a [CR].

MEDIT then asks you to verify the function. If your file was a new one, the monitor command line will say "Create Verify." If you selected a name that was already a file, it will say "Verify." If you just see "Verify," then you are warned that you are about to write over that file. If you want to go ahead with the command, type 'V.' Pressing any other key aborts the function.

PDOS errors such as an illegal filename may appear on the status line if you incorrectly name a file.

Commands used with FILE EXCERPT:
    DELETE TO POINTER ([CTRL-\])
    PLACE POINTER ([CTRL-P])

FILE INSERT ([CTRL-Y])

The FILE INSERT command allows you to insert a file directly into the text which you are currently editing. Simply place your cursor where you want the text to be inserted and type [CTRL-Y]. The editor will prompt you for the name of the file. Type in the filename which you wish to insert and terminate with a [CR]. If you typed in a filename which does not exist, the editor will display "PDOS error #53".

You will be asked to verify the command. If you wish to continue with the file insertion, type 'V' and the text will appear in your file. Press any other key if you wish to abort the FILE INSERT.

A FILE INSERT operation does not delete the file which you copied into your text.

FILE SAVE AND RETRIEVE COMMANDS

MEDIT automatically reads in a file during initialization if the filename is specified on the command line. However, it does NOT automatically save the file when you exit the editor. For your changes to be effective, you must save the file explicitly.

MEDIT saves the name of the last file retrieved or saved in a special buffer. This name can be recalled when MEDIT prompts for a filename by simply pressing FILE SAVE ([CTRL-W]) or FILE RETRIEVE ([CTRL-G]) a second time.

FILE RETRIEVE ([CTRL-G])

The FILE RETRIEVE command allows you to place files onto your screen for editing. When entering MEDIT, you can specify the name of the file you wish to edit right on the command line:

```
x>MEDIT MYFILE/O
```

Or, you can simply type 'MEDIT' and when you are in the editor, request a file with the FILE RETRIEVE command. When you type [CTRL-G], the editor asks you for the name of the file you wish to retrieve. Terminate your selection with a [CR]. You must verify the command by typing a 'V.'

The FILE RETRIEVE command clears the editor before loading a new file. To merge text from a new file with the file currently in the editor, use the FILE INSERT ([CTRL-Y]) command.

The FILE RETRIEVE command is useful when you want to edit many files without exiting the editor.

Commands used with FILE RETRIEVE:

CLEAR EDITOR ([CTRL-N])
FILE INSERT ([CTRL-Y])
FILE SAVE ([CTRL-W])

(FILE SAVE AND RETRIEVE COMMANDS continued)

FILE SAVE ([CTRL-W])

The FILE SAVE command allows you to save your work to a file. It is always a good idea to save your files periodically while you are editing. To use this command, type [CTRL-W]. The editor will prompt you for the name of the file with "Write file '". Type in the filename and terminate it with a [CR]. The editor will then prompt you with "Verify" if the file has already been defined or "Create Verify" if the file is a new one. Press 'V' to verify the command. MEDIT automatically creates new files with the FILE SAVE command if the file has not yet been defined, so there is no need to predefine files or precede the filename with a pound sign (#).

The FILE SAVE command automatically deletes the pointer and centers the screen on the cursor before writing the file out to the disk. The pointer is deleted because it occupies space in memory and, if saved to a file, would show up as garbage in the file.

If the file you wish to save has either been previously saved or was the last file entered, typing [CTRL-W][CTRL-W] will retrieve the filename. Then, you need only type a [CR] and 'V' to save the file.

Commands used with FILE SAVE:
CLEAR EDITOR ([CTRL-N])
QUIT ([ESC][CTRL-V])

FIND COMMANDS

You may search for strings forwards or backwards from your cursor with MEDIT. In either case, MEDIT saves the last string sought in a special buffer. This string can be recalled when MEDIT prompts for a search string by simply pressing ([CTRL-F]) or ([CTRL-B]) a SECOND TIME. Then hit a [CR] to complete the command.

There are a few things to keep in mind with FIND commands:

1. A [CTRL-Z] which is specified as part of the search string will be interpreted as a single-letter wild card. So, "F[CTRL-Z][CTRL-Z]D" would match both FEED and FIND.

2. A [CR] (which usually delimits the string) may be inserted in a search string by preceding it with INSERT CONTROL CHAR ([CTRL-V]). Other control characters may be inserted that same way.

3. If you want to quit in the middle of a FIND or FIND BEFORE function, type CANCEL ([CTRL-C]). You will return to normal editing mode.

FIND ([CTRL-F])

When you type FIND ([CTRL-F]), the editor will prompt you with "Find string '". Type in the text you wish to search for and end it with a [CR].

The editor will then search for the string from the point you were in your file to the end of the file. When it finds the string, the cursor will be repositioned at the end of the string.

If the string is not in the file, MEDIT prints "Not found" and the string you requested will appear on the command line. Your cursor will remain where it was.

You may also store text in the buffer with the BUFFER FILE ([CTRL-U]) command and then search for that text by entering '[CTRL-F]' (FIND), and '[ESC][CTRL-U]' (BUFFER RETRIEVE).

(FIND COMMANDS continued)

FIND BEFORE ([CTRL-B])

If you wish to search from the cursor back to the top of your file, use the FIND BEFORE ([CTRL-B]) function. It operates the same as the FIND function only in a reverse direction. The editor prompts you to type in your string with "- Find string '".

FIND AGAIN ([CTRL-A])

If you want to call up all instances of the string one by one, first use either the FIND or FIND BEFORE commands. Then, use the FIND AGAIN ([CTRL-A]) function for all successive searches. You have to do either a FIND or FIND BEFORE command before you can use FIND AGAIN.

HELP ([ESC][CTRL-A])

When you first enter the editor, the status line contains the version number of the editor and the advice "For help, enter [ESC][CTRL-A]." Simply type [ESC][CTRL-A].

Displaying the help table will not affect the file that is on your screen. When you have configured MEDIT, your commands will appear in place of the default commands. This same table also appears at the end of this section. Return to your file by pressing any key.

If your terminal screen is garbled for whatever reason, the HELP function provides a way to restore normalcy. Just press HELP ([ESC][CTRL-A]), then any key to dismiss the help screen. MEDIT will repaint the text the way it should appear.

INSERT CONTROL CHARACTER ([CTRL-V])

If you need to insert other control characters (as printer commands, etc.), you must first tell the editor to be ready to accept them or they will be ignored. The INSERT CONTROL CHARACTER function should always precede any control character that you want to place in the text. The editor will then ignore any significance that the control character may have and insert it into the file.

Inserted control characters are displayed as a single character -- the value of the character plus 32. So, a line feed or [CTRL-J] (10) is displayed as an asterisk (42) and an [ESC] or [CTRL-[] (27) is displayed as a semi-colon (59). This is only apparent on the screen display and the actual control character is stored in the file. The following table shows which characters appear on the screen to represent its control character.

CONTROL CHARACTER DISPLAY

[CTRL-A] = !	[CTRL-I] = [TAB]	[CTRL-Q] = 1	[CTRL-Y] = 9
[CTRL-B] = "	[CTRL-J] = *	[CTRL-R] = 2	[CTRL-Z] = :
[CTRL-C] = #	[CTRL-K] = +	[CTRL-S] = 3	[CTRL-[] = ;
[CTRL-D] = $	[CTRL-L] = ,	[CTRL-T] = 4	[CTRL-\] = <
[CTRL-E] = %	[CTRL-M] = [CR]	[CTRL-U] = 5	[CTRL-]] = =
[CTRL-F] = &	[CTRL-N] = .	[CTRL-V] = 6	[CTRL-^] = >
[CTRL-G] = '	[CTRL-O] = /	[CTRL-W] = 7	[CTRL-_] = ?
[CTRL-H] = (	[CTRL-P] = 0	[CTRL-X] = 8	

It is not possible to set the eighth bit on characters or to insert a null character.

Commands used with INSERT CONTROL CHARACTER:

DELETE CONTROL CHARACTERS ([ESC][CTRL-N])

INSERT TAB ([CTRL-I])

The INSERT TAB command allows you to insert a tab character (tabs are set every eight spaces) into the text. You will probably want to configure MEDIT so that the INSERT TAB command will coincide with your tab key.

There is no way to configure MEDIT to anything but eight spaces per tab stop.

JUMP

The JUMP commands are like the MOVE commands except that they cover more distance. JUMP DOWN moves the cursor multiple lines, where MOVE DOWN only moves the cursor one line, and so on.

JUMP COUNT SET ([ESC][CTRL-W])

The JUMP COUNT SET command allows you to set the jump count (the number of lines that the cursor will jump when you execute a JUMP UP OR JUMP DOWN command). The default is half of a screenful or 11 lines.

To set the jump count, type [ESC][CTRL-W]. MEDIT will display the current number of lines and allow you to enter a new number.

JUMP RIGHT ([ESC][CTRL-L])
JUMP LEFT ([ESC][CTRL-H])

To move the cursor to the end or the beginning of the line you are on, just press the escape key before selecting right or left: JUMP RIGHT ([ESC][CTRL-L]) and JUMP LEFT ([ESC][CTRL-H]).

JUMP DOWN ([ESC][CTRL-J])
JUMP UP ([ESC][CTRL-K])

JUMP DOWN ([ESC][CTRL-J]) and JUMP UP ([ESC][CTRL-K]) move the cursor down or up a certain number of lines. This number of lines can be set by you with the JUMP COUNT SET command.

When you want to move rapidly through your file by using a series of jump commands, you only need to press the escape key. Subsequent MOVE UP ([CTRL-K]) or MOVE DOWN ([CTRL-J]) commands will be interpreted as jump commands. To cancel this operation, type any key or command except MOVE UP or MOVE DOWN.

(JUMP continued)

JUMP TO TOP OF FILE ([CTRL-T])
JUMP TO BOTTOM OF FILE ([CTRL-Z])

To jump to the beginning of your file, use the JUMP TO TOP OF FILE [CTRL-T] command. To get to the end of your file, use the JUMP TO BOTTOM OF FILE [CTRL-Z] command.

JUMP TO LINE ([ESC][CTRL-G])

You can go to a particular line in the file with the JUMP TO LINE [ESC][CTRL-G] command. Enter the line number in response to the prompt "Goto line '" on the status line and terminate with a [CR]. The cursor will then move to that line.

LIST FILES ([ESC][CTRL-F])

Listing the files on your disk while you are in MEDIT is easy. When you use the LIST FILES ([ESC][CTRL-F]) command, the editor will prompt you with "List files '". You may then enter a file specification just as you would with the >LS PDOS monitor command. If you just enter a [CR], the editor will list out the directory of the current disk and level.

The directory appears on your screen just as it would from the PDOS monitor. Pressing any key except CANCEL ([CTRL-C]) or [ESC] during the listing will stop the screen. Pressing a second key will continue the listing to your screen. Pressing [CTRL-C] or [ESC] during the listing will interrupt the listing and allow you to return to your text. After you are through examining the files, you can press any key to return to the screen just as it was before.

MACROS

Macros are a series of commands that the editor performs repeatedly for you. If you want to change all instances of some text to some other text, a macro will save you a lot of work.

MACRO DEFINE ([CTRL-D])

Defining a macro with MEDIT is really very simple. First, you must decide exactly what it is you want to do and how you will achieve it. Then, type [CTRL-D]. You will know that MEDIT is recording your steps because the word "MACRO" appears in place of the clock.

Now, you can type the sequence of commands you wish to use. The editor will show you exactly what you are doing by actually performing the functions. When you have finished the sequence, press [CTRL-D] again.

A macro may have any MEDIT command in it except the following:

```
MACRO DEFINE [CTRL-D]
MACRO EXECUTE [CTRL-E]
MACRO MULTIPLE EXECUTE [ESC][CTRL-Z]
CANCEL [CTRL-C]
PDOS TYPE-AHEAD CLEAR [CTRL-X]
NULL [CTRL-@]
```

A macro may contain a MACRO RETRIEVE ([ESC][CTRL-Y]) command, but the results are unpredictable.

It is possible it define a macro that makes a modification to the file, writes it out, and exits the editor. To define this macro, you must perform all the steps (including the QUIT ([ESC][CTRL-V])). This will put you at the PDOS monitor prompt. Re-enter the editor with the >GO monitor command. Now the macro will be defined; just type MACRO SAVE ([ESC][CTRL-O]) and save it away. Such a macro allows for hands-off editing, especially if it is invoked from the PDOS monitor as shown below:

```
x>MEDIT FILENAME,,,MACRO
```

(MACROS continued)

MACRO EXECUTE ([CTRL-E])

To perform your macro again, press [CTRL-E]. The sequence of events will be displayed very rapidly on the status line, but the screen will not be updated until the macro has finished executing. Many terminals have repeating keys, and as such, you can repeatedly execute the macro by holding the [CTRL-E] keys down.

MACRO MULTIPLE EXECUTE ([ESC][CTRL-Z])

To execute your macro a number of times, press [ESC][CTRL-Z]. The editor will prompt you with "Execute macro '". Type in the number of times you wish to perform the macro and terminate the number with a [CR]. If you type in "-1," the macro will execute until the operation fails. To stop the macro after it has begun executing, type CANCEL ([CTRL-C]).

MACRO SAVE ([ESC][CTRL-O])

If you want to save the macro that you just defined to edit other files, you can write it out to a file with the MACRO SAVE ([ESC]^O) function. After you type [ESC]^O, the status line will prompt you with "Macro to file '". You type in a valid PDOS file name that you wish to call your macro and terminate your entry with a [CR]. If the filename has not been used before, the status line will then read "CREATE VERIFY." Type 'V' to complete the save.

MACRO RETRIEVE ([ESC][CTRL-Y])

To recall the macro that you have previously saved to a file, use the MACRO RETRIEVE ([ESC][CTRL-Y]) function by typing [ESC][CTRL-Y]. The editor will prompt you with "Get macro file '". All you do is type in the name of the macro you wish to retrieve, terminate with a [CR], and verify the action. If you typed in the file name of a file that does not exist, the status line will give you the message "PDOS ERROR #53." Begin the MACRO RETRIEVE function again and type in the correct name.

You may also retrieve a macro directly on the monitor command line by entering 'MEDIT FILENAME,,MACRO[CR]'. If you retrieve the macro with that method, the macro will execute once immediately.

MOVE

The cursor will be at the top left hand corner of your screen when you first arrive in MEDIT. To move in any direction one character at a time, you can use the arrows on your keyboard, or you can use the MEDIT commands. To move the cursor down one character at a time, press MOVE DOWN ([CTRL-J]). To move the cursor up one character at a time, use MOVE UP ([CTRL-K]). The MOVE RIGHT ([CTRL-L]) and MOVE LEFT ([CTRL-H]) keys are on the right and left of the MOVE UP and MOVE DOWN keys. On some terminals, the arrow keys may also work to perform the MOVE commands. If they don't, it is easy to configure the editor so that you may use your arrow keys for the MOVE commands (see 5.5 Configuring MEDIT).

See also the JUMP commands for traveling greater distances.

```
         ^
         | ^K
^H <----------> ^L
         | ^J
         v
```

MOVE LEFT -- [CTRL-H]
MOVE DOWN -- [CTRL-J]
MOVE UP -- [CTRL-K]
MOVE RIGHT -- [CTRL-L]

POINTER

The cursor always gives you one reference point in your text, but there are times when you need to have another. The pointer is another reference point that you can place anywhere in your text. Once placed in your text, the pointer is a character like any other. Just delete it with one of the DELETE commands when you no longer need it. The pointer will be deleted when you write text to a file with the FILE SAVE [(CTRL-W]) command. The screen will be updated and the text will be recentered on the cursor.

Only one pointer can be placed in the text at a time. You may wish, however, to mark multiple locations in the text; for example, when you wish to mark a position in the text, move elsewhere to mark and save a block of text, and return to the first location and insert the text you just saved. In that case, you may use any character or sequence of characters that does not normally occur in the text. For instance, you might use a [CTRL-A] as a marker by using the INSERT CONTROL CHARACTER ([CTRL-V]) command followed by [CTRL-A]. This marker appears as an exclamation point which can be found by using the FIND ([CTRL-F]) or FIND BEFORE ([CTRL-B]) command followed by [CTRL-V][CTRL-A][CR]. You must be careful to delete this marker when you are through using it.

PLACE POINTER ([CTRL-P])

The PLACE POINTER ([CTRL-P]) command is used to mark a certain spot in your text. You can then position the cursor elsewhere in the text and the pointer will not move. Deletions and insertions only affect the pointer if you delete the pointer itself.

The PLACE POINTER ([CTRL-P]) command is also used to delimit a block of text for the BUFFER SAVE ([CTRL-U]), DELETE TO POINTER ([CTRL-\]), TOGGLE UPPER/LOWER CASE ([ESC][CTRL-T]) and FILE EXCERPT ([CTRL-O]) commands.

POSITION TO POINTER ([ESC][CTRL-P])

If you would like to temporarily move to another location in your file but return to your current location, mark your present location with the PLACE POINTER ([CTRL-P]) command. Then, after you have moved elsewhere and want to return, you can find your place with the POSITION TO POINTER ([ESC][CTRL-P]) function.

QUIT ([ESC][CTRL-V])

To exit MEDIT and return to the PDOS monitor, type [ESC][CTRL-V].

MEDIT does not automatically save files when you exit. So, you must always remember to save your file before you leave the editor. If you forgot to save your file, you can get back to where you were from the PDOS monitor by typing 'GO'. You will be right back to where you were editing and then you can save the file. Don't call up MEDIT.

x>GO
Re-enters MEDIT from PDOS monitor. The file will still be in memory.

RECENTER ([CTRL-R])

The RECENTER ([CTRL-R]) function will cause the editor to rewrite the screen so that the cursor is centered on the screen. This command is useful when you need to see text before and after the location you are editing.

STATISTICS ([ESC][CTRL-B])

If you ask for STATISTICS ([ESC][CTRL-B]), the editor will display the number of free bytes, the total number of lines, and the line number on which the cursor is located. The information appears on the status line of the screen.

TOGGLE UPPER/LOWER CASE ([ESC][CTRL-T])

TOGGLE UPPER/LOWER CASE ([ESC][CTRL-T]) will switch all lower case characters to upper case and all upper case characters to lower case. Mark one end of a block of text with the PLACE POINTER ([CTRL-P]) function, move the cursor to the beginning of the block, and press [ESC][CTRL-T]. The editor will ask you to verify the command with a 'V'.

USE INSERT/REPLACE MODE

Normally MEDIT operates in INSERT MODE; i.e. text is inserted at the cursor and any existing text under and to the right of the cursor moves over. An alternate mode is REPLACE MODE where text entered from the keyboard overwrites (i.e. replaces) text under and to the right of the cursor. The letter 'R' appears by the clock when you are in replace mode. Switch back and forth between these two modes with the following commands:

USE INSERT MODE ([ESC][CTRL-I])

When you are in INSERT MODE, text entered will displace other text, pushing it to the right.

USE REPLACE MODE ([ESC][CTRL-R])

When you are in REPLACE MODE, text entered will overwrite other text. The letter 'R' appears by the clock to remind you that you are in REPLACE MODE. Replacement only occurs on the current line. Text entered on one line will not overwrite text on the following line.

5.5 CONFIGURING MEDIT

Configuring the editor allows you customize MEDIT to use function and arrow keys on your terminal. This is done by running the PDOS utility "MEDITCON" that you will find on your utility disk.

Before you run MEDITCON, it is a good idea to think out what commands you want for each function. Keep in mind which functions you use the most. A couple of things of which you should be aware are:

1. CANCEL is best left as [CTRL-C] because of its general use in PDOS as CANCEL.

2. [CTRL-X] cannot be used as it is used to clear the type-ahead buffer.

3. [CTRL-S] and [CTRL-Q] are used for handshaking on some terminals. Because PDOS supports this convention as an option, do not use these control characters as editor functions unless you will not use [CTRL-S][CTRL-Q] handshaking.

4. One command cannot be both a command prefix and a command. For instance, if your F1 key produces [CTRL-A][ESC], then you can't use [CTRL-A] as a command by itself.

Table 5.1 should be filled in before you begin.

Now, you are ready to begin by running the configurator. This is done typing MEDITCON at the PDOS monitor level.

>MEDITCON

The following message should appear on your screen though the revision number and dates may be different:

```
68000 PDOS 1.1 MEDIT Configurator
Copyright 1986 Eyring Research Institute, Inc.

The MEDITCON utility generates a new version of MEDIT which
you configure for your terminal.  Each function will appear
along with the default command for that function.  If you
want to retain the default command, enter a [CR].
```

Continued on next page...

(5.5 CONFIGURING MEDIT continued)

To change the command, press the key or series of keys you wish to use. A few seconds after your last character, MEDITCON will ask for the name of the keys you pressed. You type the name followed by a [CR]. This name will be used in the MEDIT help menu.

If you make a mistake, you can step back to the previous function by using [CTRL-C]. The [ESC] key returns you to the PDOS monitor only if pressed when naming the key.

```
ENTER NAME OF NEW EDITOR:
   ([CR] will name the new editor MEDIT)
ENTER PROCEDURE FILE NAME:
   ([CR] will use the name MEDIT:DO)
```

The configurator will now step you through a list of commands like the table you have already filled out. Your cursor is located at the beginning of the default command. If you wish to continue to use the default commands, simply press the carriage return. For instance the following will appear on your screen with your cursor located where the underline appears:

```
Buffer Fill...........[ESC]^U  (You type a [CR] to use the default)
```

After you have typed a [CR] the next command will appear as you move through the list.

```
          . . .
Move Down.............^J
```

If you wish to change MOVE DOWN to your down arrow key, simply press your down arrow key. The command sequence used by that key will then appear in place of the default command sequence. DO NOT PRESS THE CARRIAGE RETURN or MEDITCON will interpret a [CR] as part of the command sequence.

```
Move Down.............[ESC]P    (Output from the arrow key)
```

Within a few seconds, MEDITCON will ask you the name of the function key. This is the name of the key(s) that you just pressed. This information will be displayed in the MEDIT help menu.

```
Move Down............[ESC]P   Function key name?Down Arrow[CR]
```

Just type a description of the key (F1, Shift F1, Up Arrow, etc.) and terminate the entry with a carriage return. The next command in the list will then appear.

Continued on next page...

(5.5 CONFIGURING MEDIT continued)

If you wish to return to the previous command, press [CTRL-C] and you can step backwards through the command list. After you have gone through the entire list, the configurator will ask you the following question:

BUILD NEW EDITOR (Y/[N])?

If you answer 'Y', MEDITCON will chain to the procedure file it has just created (named MEDIT:DO or whatever you called it). This procedure file will run the MASM assembler on the MEDIT:DO temporary file and create a temporary object file named MEDIT:TMP. This object file will be linked to the main part of the editor, called MEDIT:OB by the PDOS linker, QLINK. The resulting program will be output as MEDIT or whatever you chose to call it.

If you answer 'N', MEDITCON will simply exit, closing the file MEDIT:DO. If you wish to create the new editor later (after, perhaps, having modified the file MEDIT:DO) you may do so by simply typing the procedure filename at the monitor prompt.

FUNCTION	DEFAULT	CONFIGURED
Buffer Fill	[CTRL-U]	________
Buffer Retrieve	[ESC][CTRL-U]	________
Cancel	[CTRL-C]	________
Clear Editor	[CTRL-N]	________
Command Mode	[ESC][CTRL-C]	________
Delete Control Chars	[ESC][CTRL-N]	________
Delete Left	[RUB]	________
Delete Line	[CTRL-^]	________
Delete Right	[CTRL-_]	________
Delete to EOL	[CTRL-]]	________
Delete to Pointer	[CTRL-\]	________
File Excerpt	[CTRL-O]	________
File Insert	[CTRL-Y]	________
File Retrieve	[CTRL-G]	________
File Save	[CTRL-W]	________
Find	[CTRL-F]	________
Find Again	[CTRL-A]	________
Find Before	[CTRL-B]	________
Help	[ESC][CTRL-A]	________
Insert Control Char	[CTRL-V]	________
Insert Tab	[CTRL-I]	________
Jump Count Set	[ESC][CTRL-W]	________
Jump Down	[ESC][CTRL-J]	________
Jump Left	[ESC][CTRL-H]	________
Jump Right	[ESC][CTRL-L]	________
Jump to Bottom of File	[CTRL-Z]	________
Jump to Line	[ESC][CTRL-G]	________
Jump to Top of File	[CTRL-T]	________
Jump Up	[ESC][CTRL-K]	________
List Files	[ESC][CTRL-F]	________
Macro Define	[CTRL-D]	________
Macro Execute	[CTRL-E]	________
Macro Multiple Execute	[ESC][CTRL-Z]	________
Macro Retrieve	[ESC][CTRL-Y]	________
Macro Save	[ESC][CTRL-O]	________
Move Down	[CTRL-J]	________
Move Left	[CTRL-H]	________
Move Right	[CTRL-L]	________
Move Up	[CTRL-K]	________
Place Pointer	[CTRL-P]	________
Position to Pointer	[ESC][CTRL-P]	________
Quit	[ESC][CTRL-V]	________
Recenter	[CTRL-R]	________
Statistics	[ESC][CTRL-B]	________
Toggle Upper/Lower Case	[ESC][CTRL-T]	________
Use Insert Mode	[ESC][CTRL-I]	________
Use Replace Mode	[ESC][CTRL-R]	________

TABLE 5.1 MEDIT FUNCTIONS

BUFFER FILL([CTRL-U])	Store from cursor to pointer in buffer
BUFFER RETRIEVE([ESC][CTRL-U])	Insert contents of buffer at cursor
CANCEL([CTRL-C])	Abort current function
CLEAR EDITOR([CTRL-N])	Clear editor workspace and all buffers
COMMAND MODE([ESC][CTRL-C])	Future feature. Currently a QUIT
DELETE CONTROL CHARS([ESC][CTRL-N])	Delete all ctrl chars & clear bit 8
DELETE LEFT([RUB])	Delete char to left of cursor
DELETE LINE([CTRL-^])	Delete right up to & including return
DELETE RIGHT([CTRL-_])	Delete char to right of cursor
DELETE TO EOL([CTRL-]])	Delete right up to return
DELETE TO POINTER([CTRL-\])	Delete back or ahead to pointer
FILE EXCERPT([CTRL-O])	Save from cursor to pointer to file
FILE INSERT([CTRL-Y])	Insert file at cursor
FILE RETRIEVE([CTRL-G])	Overwrite editor workspace with file
FILE SAVE([CTRL-W])	Write editor workspace to file
FIND([CTRL-F])	Search forward for string
FIND AGAIN([CTRL-A])	Repeat previous FIND or FIND BEFORE
FIND BEFORE([CTRL-B])	Search backward for string
HELP([ESC][CTRL-A])	Display editor functs/key assignments
INSERT CONTROL CHAR([CTRL-V])	Ignore next char's special meaning
INSERT TAB([CTRL-I])	Insert tab character at cursor
JUMP COUNT SET([ESC][CTRL-W])	Define how many lines a JUMP is
JUMP DOWN([ESC][CTRL-J])	Advance N lines forward
JUMP LEFT([ESC][CTRL-H])	Position cursor to beginning of line
JUMP RIGHT([ESC][CTRL-L])	Position cursor to end of line
JUMP TO BOTTOM OF FILE([CTRL-Z])	Position cursor to end of file
JUMP TO LINE([ESC][CTRL-G])	Position cursor to line N
JUMP TO TOP OF FILE([CTRL-T])	Position cursor to beginning of file
JUMP UP([ESC][CTRL-K])	Move cursor back N lines
LIST FILES([ESC][CTRL-F])	Display file directory on screen
MACRO DEFINE([CTRL-D])	Define sequence of commands as MACRO
MACRO EXECUTE([CTRL-E])	Execute previously defined MACRO
MACRO MULTIPLE EXECUTE([ESC][CTRL-Z])	Execute MACRO N times
MACRO RETRIEVE([ESC][CTRL-Y])	Read MACRO definition from file
MACRO SAVE([ESC][CTRL-O])	Save MACRO definition to file
MOVE DOWN([CTRL-J])	Move cursor down in same column
MOVE LEFT([CTRL-H])	Move cursor left in same row
MOVE RIGHT([CTRL-L])	Move cursor right in same row
MOVE UP([CTRL-K])	Move cursor up in same column
PLACE POINTER([CTRL-P])	Mark cursor position for future use
POSITION TO POINTER([ESC][CTRL-P])	Move to previously marked position
QUIT([ESC][CTRL-V])	Leave the editor
RECENTER([CTRL-R])	Show workspace with cursor centered
STATISTICS([ESC][CTRL-B])	Show free bytes, total & current line
TOGGLE UPPER/LOWER CASE([ESC][CTRL-T])	Convert upper to lower & lower to upper case
USE INSERT MODE([ESC][CTRL-I])	Move data over when adding text
USE REPLACE MODE([ESC][CTRL-R])	Overwrite old data when adding text

TABLE 5.2 ALPHABETICAL MEDIT COMMAND SUMMARY

CHAPTER 6

ASSEMBLE AND LINK

This chapter explains the use of the PDOS assembly development software tools. These include the 68000 assembler (MASM), 68020 assembler (MASM20), and module linker (QLINK).

(CHAPTER 11 ASSEMBLE AND LINK continued)

(CHAPTER 11 ASSEMBLE AND LINK continued)

(CHAPTER 11 ASSEMBLE AND LINK continued)

6.1 MASM 68000 ASSEMBLER

MASM is a PDOS Motorola 68000/10 assembler which runs on any 68000/10 or 68020 PDOS system. It accepts 68000/10 assembly mnemonics and directives, and outputs PDOS tagged object code and various listing files.

The assembler is a two-pass assembler. The first pass resolves all symbols. The second pass generates the object, listing, and cross reference, if selected.

Two-pass assembler

MASM can execute equally well as a background task allowing other processes, such as editing, to continue in the foreground. The assembler will notify the parent task of any errors through the message buffers when it is done.

Input and optional output files are specified by a list of file names following the MASM command or from keyboard prompts. These options are, in order:

SRC=	Assembly source file (required)
OBJ=	68000 object output file
LST=	Assembly listing file
ERR=	Assembly error file
XRF=	Symbol cross reference file

```
x>MASM TEMP:SR,T,LIST,,XREF
68000 PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=TEMP:SR
OBJ=T
LST=LIST
ERR=
XRF=XREF
END OF PASS 1
END OF PASS 2  [4 WARNINGS]
```

If n object output file is specified, MASM does not output an executable or linkable file.

6.1.1 USING THE ASSEMBLER

To use the assembler from the keyboard, insert a disk with the MASM file and enter 'MASM'. The program prompts as follows:

SRC=

The 'SRC=' prompt is for the source file name. The source file assembler symbols can be defined directly from the command line in one of two ways. First, a slash (/) following the source file name begins a symbol definition. The slash is followed by the symbol, an equal sign (=), and finally the value. This can be repeated as many times as the line length allows (80 characters).

```
x>MASM
68000 PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=MPBIOS:SR/RDZ=255/DEMO=0
OBJ=
```

Second, if a 'Q' follows the slash, then the assembler prompts on the next line for a symbol, equal sign, and value. This continues until an 'ENDQ' or [ESC] is entered.

The source file must end with an INCLUDE or END directive. The argument of the END directive is an expression whose value is output to the object file with an entry tag.

```
x>MASM
68000 PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=MBIOS:SR/Q
   *QUERY*
   :RDZ=255
   :DEMO=0
   :ENDQ
OBJ=
```

(6.1.1 USING THE ASSEMBLER continued)

OBJ=

The 'OBJ=' prompts for the object output file name. Assembler object is written to this file during pass 2 as it is assembled. This file is closed as an 'OB' file. PDOS tagged object consists of ASCII characters terminated with the character 'F' and two checksum characters. The symbol table can also be optionally output to the object file for debugging purposes using the 'OPT D' assembler directive.

OBJ=OBJ:OB

LST=

The 'LST=' prompts for a list file name. The list file is generated on pass 2 and contains paged results of the assembly process. The assembly listing is followed by a symbol table dump unless the CRE option is enabled. If CRE or cross reference has been set, then the symbol table is replaced with a cross reference of all symbols found during the assembly.

LST=LIST:TX

The list output is generated as follows:

Heading: Columns

7-8	Page number
16-20	Assembly time
23-30	Assembly date
46-120	Assembly file

Program: Columns

1-2	Line number
3-4	Error codes
5	Section
7-14	Address
16-31	Data
32	Macro id
33-40	Label field
41-48	Operation field
49-64	Operand field
65-120	Comment field

(6.1.1 USING THE ASSEMBLER continued)

ERR=

The 'ERR=' prompts for a error file name. If an error occurs during the assembly, the assembly line and an error message are output to this file. If no error file is specified, all errors are printed on your console.

```
ERR=ERR:TX
```

XRF=

The 'XRF=' prompts for a cross reference file name. After the second pass, a cross reference is output to this file.

```
XRF=XREF:TX
```

After the second pass, the error and warning count, if any, are reported. The symbol table dump consists of five parts, namely:

```
END OF PASS 1
END OF PASS 2  [1 ERROR, 4 WARNINGS]
```

```
DEFINED SYMBOLS:
EXTERNAL DEFINITIONS:
EXTERNAL REFERENCES:
UNDEFINED SYMBOLS:
UNREFERENCED SYMBOLS:
```

The symbol is followed by letter codes indicating how the symbol was generated and used. These letters are defined as follows:

```
U = Undefined              D = XDEF symbol
M = Multiply defined       L = REG list
E = EQU variable           R = Referenced symbol
S = SET variable       MACRO = Macro symbol
X = XREF symbol            I = Indirect
```

(6.1.1 USING THE ASSEMBLER continued)

An example of a program listing follows:

```
x>MASM EXAMPLE:SR,,LIST
68000 PDOS Assembler R3.2
ERII, Copyright 1983-86
SRC=EXAMPLE:SR
OBJ=OBJ
LST=LIST
ERR=
XRF=
END OF PASS 1
1/8S                       00000CF9 RL REG D0/D3-D7/A2-A3,A7
1/22w 0/0000000C:6600FFFC            BNE.L @LOOP
1/24u 0/00000012:                   END STRT
END OF PASS 2  [2 ERRORS, 1 WARNING]
x>SF LIST
```

```
                                      68000 PDOS Assembler 06-Nov-86
 PAGE: 1         11:24 10-Nov-86        FILE: EXAMPLE:SR,PDOS 3.2   SYSTEM

 1                              *      EXAMPLE:SR      12/13/83
 2                              *****************************************
 3                              *
 4           00000000                  XDEF    START
 5                                     XREF.1  MES01
 6                   00000038 LINES    EQU     56
 7                   00000050 COLN     SET     80
 8S                  00000CF9 RL       REG     D0/D3-D7/A2-A3,A7
**** ERROR 14 Syntax error
 9                   00000000 MES      EQU     MES01
10                              *
11                              OUTPUT MACRO
12                                     XPMC    &1
13                                     ENDM
14                              *
15  0/00000000:48E79F30         START  MOVEM.L RL,-(A7)
16  0/00000004:3448                    MOVEa.w A0,A2
17  0/00000006:                m       OUTPUT  MES      ;OUTPUT MESSAGE
18  0/00000006:A08C****        a       XPMC    MES
19  0/0000000A:4240                    CLR.w   D0
20                              *
21  0/0000000C:5340             @LOOP  SUBQ.W  #1,D0    ;PAUSE
22w 0/0000000E:6600FFFC                  BNE.L @LOOP
**** WARNING 23 Branch could be short  [1/8]
23  0/00000012:A00E                    XEXT             ;DONE, EXIT TO PDOS
24u 0/00000014:                        END     STRT
**** ERROR 04 Undefined symbol  [1/22]
```

(6.1.1 USING THE ASSEMBLER continued)

```
                                          68000 PDOS Assembler 06-Nov-86
PAGE: 2         11:29 10-Nov-86           FILE: EXAMPLE:SR,PDOS 3.2  SYSTEM

DEFINED SYMBOLS:

COLN      SR     00000050   LINES     ER     00000038   MES        I   MES01
MES01     X    X/00000000   OUTPUT    MACRO             RL         SL    00000CF9
START     D    0/00000000

EXTERNAL DEFINITIONS:

START     D    0/00000000

EXTERNAL REFERENCES:

MES01     X    X/00000000

UNDEFINED SYMBOLS:

STRT      UR     00000000

UNREFERENCED SYMBOLS:

COLN      SR     00000050   LINES     ER     00000038
```

6.1.2 ASSEMBLY LANGUAGE FORMAT

Assembly language source statements consist of the following four fields:

```
LABEL   OPERATION OPERANDS   COMMENT
```

```
LABEL   ADD.L   A1,A2   ;COMMENT
```

The source line must be less than 120 characters long, and at least one blank or TAB must be inserted between fields. It may also contain special first characters such as a question mark (yes or no conditional assembly -- 6.1.10.32.1), an upline (left or right conditional assembly -- 6.1.10.32.2), or a minus (inhibits listing in macro definition -- 6.1.9).

NOTE: It is assumed that the user is familiar with Motorola assembly language syntax. MASM uses a syntax modeled after that of Motorola, but the two are not completely compatible. One description of the Motorola syntax can be found in "M68000 Family Resident Structured Assembler Reference Manual" from Motorola.

LABEL FIELD

The label is a symbol consisting of one to nine characters, beginning with an alphabetic character or period in position one of the source line. The label field is terminated with a blank, TAB, colon, or carriage return. If a label is not used, character position one must be a blank or TAB character. An asterisk in position one defines a comment line.

```
LABEL
L23456789
COLON:
```

OPERATION FIELD

This field contains 1) an instruction mnemonic, 2) a directive mnemonic, 3) a macro call, or 4) a PDOS primitive. Usually this field is positioned in the second tab field, beginning eight characters from the left. Almost all four-character PDOS primitives are legal opcodes. (See Section 6.1.11.2 PDOS Primitives for a list of MASM opcodes).

68000 assembly allows extensions to the opcode mnemonics to select the instruction length. If the user does not provide the opcode length extension, MASM substitutes a lower case default extension when appropriate.

```
18 0/00000008:4240      CLR.w   D0
```

OPERAND FIELD

The operand field contains all operands of the instruction or the parameters of a macro call. When two or more operand subfields appear within a statement, they must be separated by a comma, but may not contain embedded spaces. The operands specify the addressing mode, registers, memory locations, or immediate data used by the instruction. Constants, symbols, literals, and expressions are legal operands.

(6.1.2 ASSEMBLY LANGUAGE FORMAT continued)

COMMENT FIELD

Comments follow the operand field. Usually, the comment field is positioned in the fifth tab field. The use of a semicolon as the first character in the comment field helps to set off comments for clarity and insure correct positioning with the FORMAT directive. If the first character of a source line is an asterisk (*), then the entire line is a comment. There must be a space between the operand field and the comment field.

6.1.3 CONSTANTS

Constants can be signed decimal, hexadecimal, or binary integers, ASCII constants, or 4- or 6-byte floating point numbers.

Decimal integers are written as a string of numerals in the range of -2,147,483,648 to 2,147,483,647.

```
00BC614EFFFFA48B DC.L   12345678,-23413
```

Hexadecimal constants consist of a string of one to eight hexadecimal digits, preceded by a dollar sign ($) and range from $00000000 to $FFFFFFFF.

```
80000000FFFFFFE1 DC.L   $80000000,-$1F
```

Binary constants consist of a string of 1's and 0's, preceded by a percent sign (%).

```
0000B425         DC.L   %1011010000100101
```

ASCII character constants consist of a string of from one to four characters enclosed by single quotation marks (ASCII $27). MASM generates an individual single quotation mark when two consecutive marks are encountered -- ('') -> ($27). ASCII constants are right justified. (The Motorola assembler left justifies ASCII constants.)

```
4142434400000046 DC.L   'ABCD','F'
```

6.1.4 SYMBOLS

Symbols begin with an alphabetic character or a period and can be up to nine characters in length. There can be no embedded blanks. Legal characters for positions 2 through 9 are A-Z, 0-9, ., _, and $. Lower case letters are also legal but are changed to upper case for symbol table usage.

A symbol in the operand field may be immediately followed with a '.W' or '.L' extension when using the 68000 absolute address modes. It forces either absolute short (.W) or absolute long (.L) addressing for that one operand. A global default is set with the ARS or ARL option for absolute short or long addressing, respectively.

A local symbol consists of the '@' character followed by one to four additional characters. All local symbols lose their uniqueness after a non-local label is encountered in the label field.

The assembler uses the asterisk (*) to represent the current location counter.

A particular symbol can be used in the label field only once with the exception of symbols such as SET (temporary equate) or REG (register list) variables.

The PDOS assembler supplies most system symbols required by a user. These constants are supplied on reference after the 'OPT PDOS' directive is executed and are listed in section 6.1.11.3. The following is the convention adopted for external PDOS symbols:

```
  xxx$ = TCB index (A6)                   MOVE.B   U1P$(A6),D0
  xxx. = SYRAM constant                   MULU.W   #TBZ.,D0
 xxxx. = SYRAM index (A5)                 MOVE.L   TICS.(A5),D1
  .xxx = Global system constant           MOVE.W   #.BPS,D7
 m.xxx = Module constant                  MOVE.W   #B.PTMSK,SR
 m$xxx = Module entry point               BSR.L    K2$PINT
 m_xxx = Module index                     CLR.W    B_TPS(A0)
  xxx_ = User index                       ADDA.L   AVL_(A4),A0
```

6.1.5 EXPRESSIONS OR OPERATORS

Expressions are made up of symbols and constants, each of which may be immediately preceded by a unary plus (+), 1's complement (~), or minus (-). Symbols and constants are separated by binary operators.

The binary operators interpreted by MASM for expressions are defined as follows:

=	Relational equal	000000000001	DC.W	1=2,2=1,2=2
<	Relational less than	000100000000	DC.W	1<2,2<1,2<2
<=	Relational less than or equal	000100000001	DC.W	1<=2,2<=1,2<=2
>	Relational greater than	000000010000	DC.W	1>2,2>1,2>2
>=	Relational greater than or equal	000000010001	DC.W	1>=2,2>=1,2>=2
<>	Relational not equal	000100010000	DC.W	1<>2,2<>1,2<>2
\	Modulo	000100000000	DC.W	1\2,2\1,2\2
+	Add	000300030004	DC.W	1+2,2+1,2+2
-	Subtract	FFFF00010000	DC.W	1-2,2-1,2-2
*	Multiply	000200020004	DC.W	1*2,2*1,2*2
/	Divide	000000020001	DC.W	1/2,2/1,2/2
&	Logical AND	000000000002	DC.W	1&2,2&1,2&2
!	Logical inclusive OR	000300030002	DC.W	1!2,2!1,2!2
<<	Shift left	000400040008	DC.W	1<<2,2<<1,2<<2
>>	Shift right	000000010000	DC.W	1>>2,2>>1,2>>2
-	Unary minus	FFFFFFFEFFFD	DC.W	-1,-2,-3
~	Unary 1's complement	FFFEFFFDFFFC	DC.W	~1,~2,~3

The minus sign and plus sign can be unary or binary. The tilde (~) is always unary. Expressions are evaluated using operator precedence. Parentheses can be used to change precedence. Operators of equal precedence are evaluated from left to right. Precedence is defined as follows from highest to lowest:

Parentheses
Unary
Shifts
Logical AND and OR
Multiply and Divide
Add and Subtract
Not equal and modulo
Greater than
Less than
Equal

```
00000008        DC.L    4*(5+1)/8>>2!3
```

All operations are done in 32-bit, signed arithmetic. The final results are truncated for word and byte values. Rules governing addition and subtraction of relocatable operands determine the type of the result. If the result of the expression is either absolute or simple relocatable, then MASM puts out a single value to the object file. If the result cannot be resolved to either of the above cases, MASM puts out tags and values to the object file which represent the expression in reduced form.

6.1.6 PDOS ASSEMBLY OBJECT FORMAT

The advantages of modular program development are often offset by the restrictions and weaknesses found in the relocatable object code format. In general, programs that do not require linkage are not a problem. However, subprograms that are tightly bound to other modules must follow restrictive rules dictated by the ability of the linker to resolve external references. A greater flexibility would be nice for system generation programs and libraries which involve complex object modules.

Typically, object modules contain three kinds of information: machine language code and constants, address and relocation information, and external definitions and references. The problems occur in byte relocation, PC (program counter) relative addressing, external references in arithmetic expressions, large constant blocks, split instructions, program identification, and object code transportability.

The PDOS 68000 tagged object format is very powerful and gives added flexibility to 68000 modular programming. The assembler and linker work together in the development of the final execution module.

The PDOS linker is a stack-oriented program which maintains not only a symbol table, but also expression operation lists that are used to do Reverse Polish operations at link time. Constants, symbol values, and section addresses are pushed and popped from the stack and used to build the desired object.

This means that complex, relocatable expressions can be resolved at link time. For instance, it is possible to define a constant that consists of the difference between two externally defined symbols and have the linker calculate the value.

```
         XREF A,B
C        DC.L A-B
```

6.1.6.1 68000 TAGGED OBJECT

Table 6.1 defines the object codes. The first character of each item is the key and indicates how the linker is to process subsequent characters. Items are of fixed length, except for variable length symbol names. Items are concatenated into lines of ASCII characters with a terminating checksum, to aid in transporting object files.

Tag 0 is the module identification item. It specifies how the object was generated (i.e. assembler, PASCAL, C, etc.) and gives the version and revision of the source file. The source module name or some other character identification symbol and the assembly date and time are also included.

```
PRGM1    IDNT   3.0   PDOS PRGM1
```

(6.1.6.1 68000 TAGGED OBJECT continued)

Tag 1 specifies the object entry address. This item is generated by the 'END' directive of the assembler. Its value is the directive operand and indicates the section and address of the module entry point.

```
        END     START
```

Tag 2 sets the current linker program counter to a specific section and address. This can be any absolute or relocatable section and any 32-bit address.

```
        SECTION 0
        RORG    $1000
```

Tags 3, 4, and 5 store absolute data in the linked output stream. The tags correspond to bytes, words, and long words, respectively.

```
        DC.B    3
        DC.W    4
        DC.L    5
```

The linker maintains a stack which is used for all arithmetic and shift operations. The top item on the stack can be popped, when required, into the output stream. Tags 6, 7, and 8 pop a byte, word, or long word respectively.

```
        XREF    B,W,L
        MOVE.B  B(A0,D0.W),D0
        LEA.L   W(A5),A0
        MOVE.L  #L,D0
```

Tag 9 pushes an absolute or relocatable constant on the stack. A space character following the 9 tag indicates that the 32-bit value is an absolute constant. Likewise, hex characters 0 through F are for sections 0 through 15.

```
        XREF    X
        MOVE.W  #X-10,D0
```

Tag A pushes a symbol value on the stack. The tag is followed by the symbol section. If the section is a space character, then the first symbol match in the linker symbol table is used. The variable length symbol name follows. A length character precedes the symbol. These items are generated by the 'XREF' directive of the assembler.

```
        XREF    X
SYM     MOVE.W  #X-SYM,D0
```

Tag B directs the linker to do some operation using stack values. The binary operations available are add, subtract, multiply, divide, AND, OR, shift right, and shift left. The unary negate operator is also available. All operations pop the long word operands, perform the operation, and push the long word result back on the stack.

```
        XREF    X
        DC.W    (X-10)<<3+$10
```

Tag C places an external definition in the linker symbol table. The tag is followed by the symbol section. If the section is a space character, then the symbol is absolute. The variable length symbol name follows with a length character preceding the symbol. The final parameter is the 32-bit symbol value. These items are generated by the 'XDEF' directive of the assembler.

```
        XDEF    SYM
SYM     EQU     *
```

Tag D is for multiple word stores. Up to 65536 2-byte constants are stored with a single item.

```
        DCB.W   20,'  '
```

Tag E informs the linker of the length of each section contained within the module. This information allows the linker to group sections together as one section.

(6.1.6.1 68000 TAGGED OBJECT continued)

Finally, tag F is the end-of-record tag and is followed by two 2's complement checksum characters. This helps maintain data integrity and object transportation through RS232 networks. The checksum is calculated such that when added to the sum of the preceding characters in the line (included the 'F' tag), the result is zero.

A sample code to generate the checksum follows. Address register A3 is pointing to the end of the object while address register A2 points to the beginning of the record.

```
CKSM    MOVE.B  #'F',(A3)+      ;TERMINATE LINE
        CLR.B   (A3)
        CLR.L   D1              ;CLEAR CHECKSUM
        MOVEA.L A2,A1           ;POINT TO LIST
*
@0002   ADD.B   (A1)+,D1        ;CHECKSUM LINE
        TST.B   (A1)            ;DONE?
          BNE.S @0002           ;N
        NEG.B   D1              ;Y, NEGATE CHECKSUM
        XCBH                    ;CONVERT
        ADDQ.W  #6,A1
        MOVE.B  (A1)+,(A3)+     ;INSERT CHECKSUM CHARACTERS
        MOVE.B  (A1),(A3)+
        MOVE.B  #$0A,(A3)+
        MOVE.B  #$0D,(A3)+
        CLR.B   (A3)            ;TERMINATE LINE
        RTS
```

The IDNT and E tags are output to the object file after the first pass of the assembler. When the 'END' is encountered on the second pass, a final record is output to the object file that includes the assembler revision and the current date and time.

Example:

```
....
C 4SEC.00000022C 4YRS.0000001AC 5CKSM.000000C4FB6
:MASM R3.2  27-Oct-86 13:53:16
```

```
            Definition   Tag/Syntax
 ---------------------   --------------------------
 Module identification - OT--LABEL--rrrvvvddddddtttt
           Entry point - 1Saaaaaaaa
               Address - 2Saaaaaaaa
      Simple data byte - 3nn
      Simple data word - 4nnnn
 Simple long data word - 5nnnnnnnn
              Pop byte - 6
              Pop word - 7
         Pop long word - 8
            Push value - 9Snnnnnnnn
           Push symbol - ASl<symbol>
        Link operation - BO
   External definition - CSl<symbol>nnnnnnnn
   Store multiple word - Dccccnnnn
        Section length - ESnnnnnnnn
         End of record - Fcc

 Where:  r=Revision   a=Address
         v=Version    n=Hex data
         d=Date       l=Length
         t=Time       c=Count
         S=Section   cc=Checksum

         O=Operations:   0=Add        5=OR
                         1=Subtract   6=Shift left
                         2=Multiply   7=Shift right
                         3=Divide     8=Negate
                         4=AND        9=NOT

         T=Type:         A=Assembler  F=Fortran
                         B=BASIC      P=Pascal
                         C='C'

 PDOS 68000 tagged object uses a stack at link time
 to resolve external references and perform external
 calculations.
```

TABLE 6.1 PDOS 68000 TAGGED OBJECT.

6.1.6.2 AN EXAMPLE

Consider the assembly example in table 6.2. To correctly resolve these assembly statements requires byte relocation, external arithmetic and shifts, and program counter relative resolution. All are easily and efficiently handled by the tagged object format.

Source listing:

```
        XREF.1  TBL,Y,2:Z
        MOVE.B  TBL(PC,D1.W),(Y+Z)>>2(A2)
        END
```

Assembler listing:

```
1                            XREF.1 TBL,Y,2:Z
2  0/00000000:157B10******   MOVE.B TBL(PC,D1.W),(Y+Z)>>2(A2)
3  0/00000006:               END
```

Object listing:

```
E0000000064157BA 3TBL9000000002B13106A 1YA21ZB09 00000002B77F28
:MASM R3.2 11/1/86 15:51:57
```

A 68000 assembly example shows the resulting tagged object from PC relative, byte relocation, and external displacement arithmetic.

TABLE 6.2 SAMPLE TAGGED OBJECT

Assuming the following values, the link process proceeds as follows:

```
Section 0 base = $000001000
Section 2 base = $000002000
         0:TBL = $000000040
             Y = $000000064
           2:Z = $000000300
```

Continued on next page. . .

(6.1.6.2 AN EXAMPLE continued)

```
                                                                Top of Stack
 1. E000000006 = Declare section 0 to be 6 bytes long.          ------------
 2.      4157B = Output simple data word $157B.
 3.     A 3TBL = Lookup "TBL", add section 0, and push value.   ($00001040)
 4. 9000000002 = Push section 0 + $00000002.                    ($00001002)
 5.         B1 = Pop operands, subtract, and push result.       ($0000003E)
 6.        310 = Output simple data byte $10.                   ($0000003E)
 7.          6 = Pop byte ($3E) and output.
 8.       A 1Y = Lookup "Y" and push value.                     ($00000064)
 9.       A21Z = Lookup "2:Z", add section 2, and push value.   ($00002300)
10.         B0 = Pop operands, add, and push result.            ($00002364)
11. 9 00000002 = Push absolute constant $00000002.              ($00000002)
12.         B7 = Pop operands, shift right, and push result.    ($000008D9)
13.          7 = Pop word ($08D9) and output.
14.        F28 = End of line, check checksum.
```

The resulting output object stream:

```
157B 10 3E 08D9
```

would be loaded at memory location $00001000.

6.1.6.3 MASM AND QLINK

QLINK is the PDOS linker utility. When used in conjunction with the PDOS assembler, program modules are bound together into an executable module.

Table 6.3 is a listing of the QLINK map after linking the object from the example program listed in Table 6.2 and the object resulting from the following short program:

```
         XDEF     Y,Z
Y        EQU      100       ;Y = ABSOLUTE 100
         SECTION  2
         DS.B     $300
Z        EQU      *         ;Z = $300 BIASED BY SECTION 2
         END
```

Where possible, all external references have been resolved. All external definitions are listed with their value, section value, and references. The Reverse Polish equations required to resolve a particular reference are listed along with the resolved values. Other information includes grouped and ignored sections, references that overflow, and a list of all unresolved references.

```
USER ALIASES: NONE
MEMORY BUFFER BASE ADDRESS=00000000

EXTERNALLY DEFINED SYMBOLS:
NAME          MODULE       VALUE    S/DISPL     REFERENCES
TBL       U                00000000             #1 0/00000002
Y             OBJ1         00000064             #1 0/00000004
Z             OBJ1         00002300 2/00000300  #1 0/00000004

INPUT FILE MAP:
INDEX FILE NAME  TYP IDNT  R V  DATE TIME  SECTION ADDRESSES
  1   OBJ/8                                0/00000000 00000005
  2   OBJ1/8                               2/00000000 000002FF

SECTION GROUPS: NONE
IGNORED SECTIONS: NONE
OVERFLOW REFERENCE VALUES: NONE

XREF OPERATION LIST:
ADDRESS           VALUE
0/00000003.B :=  TBL 0/00000002 -
0/00000004.W :=  Y 2/Z + 00000002 >>

SECTION        BASE        LOWEST      HIGHEST
   0         00001000     00001000     00001006
   2         00002000     00002000     00002300

UNRESOLVED EXTERNAL DEFINITIONS:
NAME          MODULE       VALUE    S/DISPL     REFERENCES
TBL       U                00000000

UNRESOLVED EXTERNAL REFERENCES:
ADDRESS           VALUE
0/00000003.B :=  TBL 0/00000002 -

RESOLVED REFERENCE VALUES:
ADDRESS        VALUE
00001003.B := UNRESOLVED  00001004.W := 000008D9
```

The QLINK link map lists all input object modules, how each external reference was resolved, the resulting section addresses, and all unresolved references.

TABLE 6.3 QLINK MAP

6.1.7 ASSEMBLER ERROR DEFINITIONS

Assembler diagnostics are divided into warnings and errors. Errors cause the assembler to replace the object with illegal object code and notify PDOS through the error register (LEN$(A6)) with the error number (300-399). Warnings simply indicate that something might be amiss but do not affect the object and can be disabled with the NOWARN parameter in an OPT directive.

Code	Warning	Number/Description	
c	Yes	300 Modified instruction	Altered warning
s		301 Illegal symbol	Symbol errors
M		302 Multiply defined symbol	
m	Yes	303 Multiply defined symbol referenced	
u		304 Undefined symbol	
p		305 Phase error	
x		306 Illegal opcode	Opcode and operand errors
e		307 Illegal opcode extension	
b	Yes	308 Was on odd byte boundary	
o		309 Missing operand	
i		310 Illegal operand mode	
-		311 Unary operator error	Evaluation errors
U		312 Stack underflow	
O		313 Stack overflow	
S		314 Syntax error	
A		315 Absolute expression required	
W		316 Illegal complex expression	
a	Yes	317 Arithmetic overflow	
n	Yes	318 Numeric overflow	
d		319 Displacement field overflow	
z		320 Division by zero	
'	Yes	321 Unmatched quotes or parens	
B		322 Branch to odd address	Parameter errors
w	Yes	323 Branch could be shorter	
r		324 Parameter out of range	
L		325 Illegal register list	
t	Yes	326 String truncated	
X		327 Illegal section specification	
P		328 Illegal OPTION	
l		329 Label not allowed	Assembler context errors
C		330 IF/ENDC or MACRO/ENDM error	
f		331 Floating point error	

(6.1.7 ASSEMBLER ERROR DEFINITIONS continued)

After the assembly, if errors occurred and there is no input port assigned to the task, then the error report is sent to the parent task through the message buffers.

Auxiliary errors are additional information for diagnosing an assembler error. They are generally associated with conditional assembly or macros.

Error	Description
332	ENDC w/out matching IFxx
333	ENDM w/out MACRO header
334	Legal only in body of macro
335	Macro label not found
336	Must be symbol
337	Label required
338	Macro definitions cannot be nested
339	Infinite parameter substitution

6.1.8 ASSEMBLER DEFINITIONS AND DEFAULTS

The following are predefined mnemonic symbols that are recognized by the assembler:

D0-D7	Data registers
A0-A7	Address registers
A7, SP	Stack pointer
USP	User stack pointer
CCR	Condition code register
SR	Status register
PC	Program counter
*	Current location counter

The standard version of MASM uses the following default values. Contact Eyring for pricing and delivery of custom versions of MASM with different parameters from those listed. Only the number of lines per page (NLP) can be dynamically altered by the user. Use the OPT directive NLP=# (section 6.1.10.32.18).

NLP = 56	Number of lines/page
NMC = 8	Number of nested macros
LLN = 120	Maximum for LLEN
ITZ = 80	Maximum item size
OBS = 60	Output object line size
BLN = 3*8	Output object debug length
ELZ = 80	Error list size
DBZ = 80	Debug buffer size

6.1.9 ASSEMBLER MACROS

Assembler macros provide line replacement with parameter substitution. The macro is defined with the MACRO directive in the operation field; the symbol in the label field is the macro name. The body of the macro follows and is terminated with the ENDM directive. All lines between the MACRO and ENDM directives are saved.

A macro is called whenever the macro name appears in the operation field. The subfields of the operand field are assigned as the parameters used during the macro expansion. These parameters are referenced as numbers one through nine and are global when calling macros within macros (nesting).

Note: This is the major difference between the PDOS and Motorola macros. Motorola keeps parameters local when nesting them.

Parameter substitution is signaled by the ampersand (&) character followed by a number. Parameters &1 through &9 are replaced by operands 1 through 9 respectively. If no operand was specified for a particular parameter, then nothing is substituted.

```
                            ************************
                            * PARAMETER SUBSTITUTION
                            ************************
                            *
                            MAC1 MACRO
                                 DC.B    &1,&2
                                 ENDM
                            *
   0/00000000:         m         MAC1    10*2,-1
   0/00000000:14FF     a         DC.B    10*2,-1
```

The parameter &# is replaced with the ASCII decimal equivalent of a macro counter. The counter starts at zero and is incremented by one whenever a macro is expanded. This means that the counter is equal to 1 if it is referenced in the first macro call.

```
                            ************************
                            * MACRO COUNTER
                            ************************
                            *
                            MAC2 MACRO
                            LB&# DC.B    &1,&2
                                 ENDM
                            *
   0/00000002:         m         MAC2    3,2
   0/00000002:0302     aLB2 DC.B    3,2
```

The macro parameter &0 is replaced by the macro extension characters appended to the macro name if any. This includes the period. Legal extensions are ".S", ".B", ".W", and ".L".

```
                            ************************
                            * MACRO EXTENSION
                            ************************
                            *
                            MAC3 MACRO
                                 DC&0    &1
                                 ENDM
                            *
   0/00000004:         m         MAC3.W  10
   0/00000004:000A     a         DC.W    10
```

(6.1.9 ASSEMBLER MACROS continued)

The characters &* are replaced within the macro expansion by the complete macro call line. This is useful in error reporting during macro expansion.

```
                        ************************
                        * MACRO HEADER
                        ************************
                        *
                        MAC4 MACRO
                             DC.B    '&*'
                             ENDM
                        *
0/00000006:           m MAC4 1,2
0/00000006:204D a            DC.B    ' MAC4 1,2'
           4143 a
           3420 a
           312C a
           32   a
```

The characters &@ are replaced with the number of parameters passed to the macro in the parameter list. The value may be used to conditionalize macros where different numbers of parameters are passed. It corresponds to the NARG symbol of the Motorola assembler.

```
                        ************************
                        * MACRO CONDITIONALIZE
                        ************************
                        *
                        . . .
                       IFEQ        &@-2
                       DC.L        &1,&2
                       MEXIT
                       ENDC
                        . . .
```

An ampersand in a macro body is inserted by a double ampersand (&&). Otherwise, the expansion looks for some other character substitution.

```
                        ************************
                        * AMPERSAND
                        ************************
                        *
                        MAC5 MACRO
                             DC.B    &1&&2
                             ENDM
                        *
0/0000000F:           m      MAC5    10,$0F
0/0000000F:0A         a      DC.B    10&$0F
```

Symbol values are substituted in a program line by enclosing the symbol between ampersands. The decimal equivalent of the symbol value replaces the ampersands and symbol name.

```
                        ************************
                        * SYMBOL SUBSTITUTION
                        ************************
                        *
                        MAC6 MACRO
                        -I   SET     10
                             DC.W    &I&*&1
                             ENDM
                        *
0/0000000F:           m      MAC6    5
0/0000000F:0032 a            DC.W    10*5
```

(6.1.9 ASSEMBLER MACROS continued)

Parameter substrings are inserted by selecting a starting character and character count within braces. These are followed by the parameter number.

```
                           ************************
                           * PARAMETER SUBSTRING
                           ************************
                           *
                           MAC7 MACRO
                                DC.B     '&{4,2}1'
                                ENDM
                           *
 0/00000011:           m        MAC7     ABCDEFGHIJK
 0/00000011:4445 a              DC.B     'DE'
```

Parameters may be dynamically selected during macro expansion by enclosing a symbol within &(and)&. The symbol is evaluated and the result used to select the desired parameter.

```
                           ************************
                           * PARAMETER SELECTION
                           ************************
                           *
                           MAC8 MACRO
                           -I   SET      2
                                DC.W     &(I)&
                                ENDM
                           *
 0/00000013:           m        MAC8     10,20,30,40
 0/00000013:0014 a              DC.W     20
```

(6.1.9 ASSEMBLER MACROS continued)

Loops in a macro expansion are done with the MIF, MIFxx, and MGOTO directives. MGOTO has only a label argument while MIF and MIFxx have two arguments. The first argument is an expression and the second is a macro label. A macro label is any character string. A macro label is placed in the code by preceding the label with an asterisk (thus making it a comment to the assembler).

```
 1                          ************************
 2                          * CONDITIONAL LOOPING
 3                          ************************
 4                          *
 5                          MAC9 MACRO
 6                          -I   SET     0
 7                          -*NXT
 8                          -I   SET     I+1
 9                          -    MIF     I>&@,END
10                               DC.W    &(I)&
11                          -    MGOTO   NXT
12                          -*END
13                               ENDM
14                          *
15 0/00000015:          m        MAC9    10,20,30,40
16 0/00000015:000A a             DC.W    10
17 0/00000017:0014 a             DC.W    20
18 0/00000019:001E a             DC.W    30
19 0/0000001B:0028 a             DC.W    40
```

Expressions can be pushed on a macro parameter stack with the MPUSH directive. Likewise, values are popped from the stack into symbols with the MPOP directive. A macro expansion can be aborted with the MEXIT directive.

Macro lines will not be printed in the expansion if the line is preceded with a minus sign.

The macro header is indicated by an 'm' character in column 32 of the list line. The body of the macro is likewise indicated with an 'a' character.

Macros may be nested 8 deep.

```
 1                          ************************
 2                          * RECURSIVE MACROS
 3                          ************************
 4                          *
 5                          FACT MACRO
 6                          -    MIFNE   &1>1,NXT
 7                          -I   SET     1
 8                               DC&0    &I&
 9                          -    MEXIT
10                          -*NXT
11                          -    MPUSH   &1
12                          -I   SET     &1-1
13                          -    FACT&0  &I&
14                          -    MPOP    II
15                          -I   SET     &I&*II
16                               DC&0    &I&
17                               ENDM
18                          *
19 0/0000001D:          m        FACT.B  2
20 0/0000001D:01   b             DC.B    1
21 0/0000001E:02   a             DC.B    2
22 0/0000001F:          m        FACT.W  6
23 0/0000001F:0001 f             DC.W    1
24 0/00000021:0002 e             DC.W    2
25 0/00000023:0006 d             DC.W    6
26 0/00000025:0018 c             DC.W    24
27 0/00000027:0078 b             DC.W    120
28 0/00000029:02D0 a             DC.W    720
```

6.1.10 ASSEMBLER DIRECTIVES

The PDOS MASM assembler supports the following directives:

ASSEMBLY CONTROL

END	End assembly
ENDC	End conditional assembly
ENDM	End macro definition
IFxx	Conditional assembly
IFDEF	Execute if defined
IFUDF	Execute if undefined
INCLUDE	Include file
MACRO	Macro definition
MEXIT	Exit macro
MGOTO	Macro GOTO
MIFxx	Macro conditional GOTO
MPOP	Macro pop from stack
MPUSH	Macro push to stack
OFFSET	Define offsets
ORG	Absolute origin
RORG	Relocatable PC adjust
SECTION	Relocatable program section

SYMBOL DEFINITION

EQU	Define assembly constant
REG	Define register list
SET	Redefine assembly constant

DATA DEFINITION

DC	Define constant
DCB	Define constant block
DCE	Define encoded string constant
DS	Define storage
EVEN	Set word boundary

(6.1.10 ASSEMBLER DIRECTIVES continued)

LISTING CONTROL AND OPTIONS

FAIL	Output fail string
FORMAT	Format listing
LIST	Enable output to list file
LLEN	Set list output line length
NOFORMAT	No list formatting
NOLIST or NOL	No output to list file
NOOBJ	No output to object file
NOPAGE	No automatic paging
OBJ	Enable output to object file
OPT	Assembler options
PAGE	Top of page
PRINT	Print to console
SPC	Space between source lines
TTL	Title

LINKER CONTROL

EXTN	External symbol
IDNT	Program identification
XDEF	External symbol definition
XREF	External symbol reference

6.1.10.1 DC - DEFINE CONSTANT

Format: [<label>] DC[.qualifier] <expression>[,...] [<comment>]

The DC directive defines a constant in memory. It may have one or more operands which are separated by commas. The qualifier specifies the storage type, where ".B", ".W", and ".L defines a byte, word, or long word. The default size is word (.W).

If the operand is a string enclosed by single quotation marks, then a byte ASCII memory allocation results.

The reserved words $DATE and $TIME are translated to ASCII strings of the system date and system time.

*Note: The DC directive does not align word and long word constants on even addresses.

```
DC.L    1,2,3,4

DC.B    $0A,$0D,'HELLO',0

DC.B    'DATE=',$DATE,0

DC.B    'TIME=',$TIME,0
```

6.1.10.2 DCB - DEFINE CONSTANT BLOCK

Format: [<label>] DCB[.qualifier] <length>,<value> [<comment>]

The DCB directive causes the assembler to allocate a block of bytes (.B), words (.W), or long words (.L), depending upon the qualifier. If the qualifier is omitted, word (.W) is the default size. The block length is specified by the absolute expression <length> and the value by <value>. <Length> can range from 0 to 32767.

```
DCB.B   20,' '
```

6.1.10.3 DCE - DEFINE ENCODED STRING

Format: [<label>] DCE.B <string or expression>[,...] [<comment>]

The DCE directive is similar to the DC.B directive. However, whenever possible, string constants have single spaces encoded by negating the previous character and multiple spaces replaced with a negative space count. Such strings are compatible with the PDOS primitives XPEL (put encoded line) and XPEM (put encoded message).

```
DCE.B   $80,'ENCODED STRING',0
```

6.1.10.4 DS - DEFINE STORAGE

Format: [<label>] DS[.qualifier] <expression>[,...] [<comment>]

The DS directive reserves memory location. The contents of the memory reserved are not initialized in any way. The <label> references the lowest address of the defined storage area. The number of bytes, words, or long words is specified in <expressions> which must be absolute and contain no forward, undefined, or external references. The qualifier specifies the storage type, where ".B", ".W", and ".L defines a byte, word, or long word. The default size is word (.W).

```
TEMP    DS.L    1
        DS.W    0       ;EVEN
```

6.1.10.5 END - END ASSEMBLY

Format: [<label>] END [<start address>] [<comment>]

The END directive informs the assembler that the source is finished. Subsequent source statements are ignored. The value of <start address>, if given, is output with a start tag in the object.

```
        RTS             ;RETURN
*
        END     START   ;END-OF-PROGRAM
```

After the second pass, the assembler name, revision, version, date, and time are output to the object file.

```
:MASM R3.2  27-Oct-86 13:53:16
```

6.1.10.6 ENDC - END CONDITIONAL ASSEMBLY

Format: ENDC

The ENDC directive terminates a conditional assembly block. Since blocks may be nested, the ENDC applies only to the last IFxx directive header.

```
        IFNE    DFLG
        DS.L    DFLG
        ENDC
```

6.1.10.7 ENDM - END MACRO DEFINITION

Format: ENDM

The ENDM directive terminates a macro definition.

```
NODE    MACRO
        MOVEA.L AVAIL(A6),A0
        SUBA.W  #&1,A0
        MOVEA.L A0,AVAIL(A6)
        ENDM
```

6.1.10.8 EQU - DEFINE ASSEMBLY CONSTANT

Format: [<label>] EQU <expression> [<comments>]

The EQU directive assigns the value of the expression in the operand field to the symbol in the label field. The <label> is optional. A well-defined expression is not required on the first pass.

```
TPS     EQU     100
```

<Label> may be equated to an external symbol thus assuming all its attributes. This is termed an indirect symbol.

6.1.10.9 EVEN - SET WORD BOUNDARY

Format: [<label>] EVEN [<comment>]

The EVEN directive forces a word alignment. A single byte of storage is allocated if the current program counter is odd.

```
TEMP    DS.L    1
        EVEN
```

6.1.10.10 EXTN - EXTERNAL SYMBOL

Format: [<label>] EXTN <symbol>[,<symbol>....

The EXTN directive declares the specified symbols to be either externally defined or referenced depending upon how they were defined by the assembler.

```
        EXTN    K$MASK
```

If the EXTN symbol is defined on the second pass, then the symbol and value are passed on to the linker as symbols which may be referenced by other modules linked to the current module. (See XDEF.)

If the EXTN symbol is undefined at the end of the first pass, then at the start of pass two, the symbol is declared as an external reference to be defined later by the linker in another module. (See XREF.)

6.1.10.11 FAIL - OUTPUT FAIL STRING

Format: [<label>] FAIL <string>

The FAIL directive outputs the <string> to your console each time it is encountered. The entire source line from the operand field is printed. The assembler loads the error register with error 67, parameter error.

```
IFLT    PTMSK<SYMSK
FAIL    **ERROR - PTMSK < SYMSK
ENDC
```

6.1.10.12 FORMAT - FORMAT LISTING

Format: FORMAT {<c#1>},{<c#2>},{<c#3>},{<c#4>} [<comment>]

The FORMAT directive formats the source list to column alignments as specified by the four parameters. Columns are numbered with 0 signifying the leftmost column of the source code area of the listing line.

<c#1> specifies the column number of the first character of the label field, <c#2> the first column of the operation field, <c#3> the operand field, and <c#4> the comment field. The columns must be increasing (you can't swap field positions). The defaults are 0,8,16, and 32. The NOFORMAT directive disables source field formatting. (See NOFORMAT.)

```
FORMAT  0,12,20,36        ;allow long labels
. . .
NOFORMAT
  or
FORMAT  0,8,16,32         ;return to defaults
```

6.1.10.13 IFDEF - EXECUTE IF DEFINED

Format: IFDEF <symbol> :<statement>

The IFDEF directive assembles the <statement> following the colon only if the <symbol> is defined. The <symbol> must be separated from the colon (:) by either a blank or tab.

```
IFDEF   B$MAP : BSR.L   B$MAP
```

6.1.10.14 IFUDF - EXECUTE IF UNDEFINED

Format: IFUDF <symbol> :<statement>

The IFUDF directive assembles the <statement> following the colon only if the <symbol> is undefined. The <symbol> must be separated from the colon (:) by either a blank or tab.

```
IFUDF   TPS     :TPS EQU 100
```

6.1.10.15 IFxx - CONDITIONAL ASSEMBLY

```
Format:         IFxx    <absolute expression>
                IFC     '<string1>','<string2>'
                IFNC    '<string1>','<string2>'
```

The IFxx directives conditionally select blocks of source code to be assembled. If the condition is TRUE, then the block is assembled. If the condition is FALSE, the block is skipped and the assembly process continues after the corresponding ENDC statement.

```
        IFLT    PTMSK<SYMSK
        FAIL    **ERROR - PTMSK < SYMSK
        ENDC
```

Valid directives for expressions are defined as follows:

```
IFEQ    If <expression> = 0
IFGE    If <expression> >= 0
IFGT    If <expression> > 0
IFLE    If <expression> <= 0
IFLT    If <expression> < 0
IFNE    If <expression> <> 0
```

Valid directives for string comparisons are defined as follows:

```
IFC     If <string1> = <string2>
IFNC    If <string1> <> <string2>
```

```
IFC 'CHAIN',&1
  (do CHAIN)
ENDC
IFNC 'CHAIN',&1
  (do NOCHAIN)
ENDC
```

If strings are not enclosed in single quotation marks, then macro parameters are changed to upper case. Quotation marks are included in the string comparison. Conditional assembly blocks can be nested up to 4 deep.

6.1.10.16 IDNT - PROGRAM IDENTIFICATION

```
Format: [<label>]  IDNT  <revision>.<version>  [<comment>]
```

The IDNT directive outputs a program identification object record to the object file. This includes a label, revision and version number, as well as the current date and time.

```
MPDOSK  IDNT    3.2     PDOS KERNEL
```

The resulting object record is defined as follows:

```
OT--LABEL--rrrvvvddddddtttt
  \\        \  \  \    \  \___
   \\        \  \  \    \_____ Time
    \\        \  \  \_________ Date
     \\        \  \___________ Version
      \\        \_____________ Revision
       \\_____________________ Label of file name
        \_____________________ Object type
```

If no <label> is given, then the field value is the source name. The object type is declared in the 'OPT Tx' option. The default is 'A' for assembly.

6.1.10.17 INCLUDE - INCLUDE FILE

Format: INCLUDE <filename>

The INCLUDE directive inserts a new source file specified by <filename>, into the assembly list. Includes may be nested up to 4 levels. Lower case file names are allowed if the file name is enclosed in single quotation marks.

```
        INCLUDE MASM1:SR
        INCLUDE MASM2:SR
        END     MASM
```

6.1.10.18 LIST - ENABLE OUTPUT TO LIST FILE

Format: LIST

The LIST directive enables assembly listing to the output file. This option is selected by default. Source text continues to be printed until a NOLIST or END directive is encountered. (See NOLIST.)

```
        LIST
```

6.1.10.19 LLEN - SET LIST LINE LENGTH

Format: LLEN <expression>

The LLEN directive sets the number of columns output for each line to the LIST file. The maximum is 120 columns.

```
        LLEN    80
```

6.1.10.20 MACRO - MACRO DEFINITION

Format: <label> MACRO [<comment>]

The MACRO directive begins the definition of an assembler macro. The <label> becomes the name of the macro. The body of the macro is terminated with the ENDM statement.

Macros may be nested 8 deep.

```
DECA    MACRO           ;DECREMENT ADDRESS
        SUBA.W  #&1,&2
        ENDM
*
        DECA    10,A0
```

6.1.10.21 MEXIT - EXIT MACRO

Format: MEXIT [<comments>]

The MEXIT directive terminates expansion of the current macro call. It is legal only within a macro definition.

```
TABLE   MACRO
I       SET     &1
*AGAIN
        MIFNE   I,NEXT
        MEXIT
*NEXT

        DC.W    I
I       SET     I+1
        MIFEQ   0,AGAIN
        ENDM
```

6.1.10.22 MGOTO - MACRO GOTO

Format: MGOTO <macro label>

The MGOTO directive transfers macro expansion to a new point within the macro as specified by <label>. A macro label can be any symbol and is preceded by an asterisk (*) at the destination line.

```
DATA MACRO
-I   SET     0
-*NXT
-I   SET     I+1
-    MIF     I>&@,END
     DC.W    &(I)&
-    MGOTO   NXT

-*END
     ENDM
```

6.1.10.23 MIFxx - MACRO CONDITIONAL GOTO

Format: MIF <abs expression>,<macro label>
MIFEQ <abs expression>,<macro label>
MIFNE <abs expression>,<macro label>

The MIFxx directives are used within macro expansions to transfer to a new point within the macro. A macro label can be any symbol and is preceded by an asterisk (*) at the destination line. MIF and MIFNE make the transfer if <abs expression> is nonzero; MIFEQ transfers if <abs expression> is zero.

```
TAB1    MACRO
I       SET     &1
*LOOP
        DC.L    I
I       SET     I-1
        MIFNE   I>0,LOOP
        ENDM
```

6.1.10.24 MPOP - POP FROM MACRO STACK

Format: MPOP <symbol>[,<symbol>...]

The MPOP directive pops a 32-bit long value from the macro stack into a symbol.

```
MPUSH   1,2,3
MPOP    A,B,C
ENDM
```

6.1.10.25 MPUSH - PUSH TO MACRO STACK

Format: MPUSH <exp>[,<exp>...]

The MPUSH directive pushes the results of each operand expression to the macro stack.

```
MPUSH   10*2,I,&2
ENDM
```

6.1.10.26 NOFORMAT - NO LIST FORMATTING

Format: NOFORMAT

The NOFORMAT option disables any further automatic formatting of source lines. (See FORMAT.)

```
NOFORMAT
```

6.1.10.27 NOLIST or NOL - NO LIST TO FILE

Format: NOLIST
NOL

The NOLIST directive disables output to the LIST file until a LIST directive is encountered.

```
NOL
```

6.1.10.28 NOOBJ - NO OUTPUT TO OBJECT FILE

Format: NOOBJ

The NOOBJ directive suppresses any further object code output to the object file until an OBJ directive is encountered.

```
NOOBJ
```

6.1.10.29 NOPAGE - NO AUTOMATIC PAGING

Format: NOPAGE

The NOPAGE directive discontinues any further paging of the listing output. Lines are printed continuously with no page headings or top and bottom margins. No label or operand is allowed, and no machine code results. Normal paging is re-enabled if a PAGE directive is encountered.

```
        NOPAGE
```

6.1.10.30 OBJ - ENABLE OBJECT FILE OUTPUT

Format: OBJ

The OBJ directive enables object output to the object file. This is the default option and continues until a NOOBJ or END statement is encountered.

```
        OBJ
```

6.1.10.31 OFFSET - DEFINE OFFSETS

Format: [<label>] OFFSET [<expression>] [<comments>]

The OFFSET directive is used to define a table of offsets via the Define Storage (DS) directive without passing these storage definitions to the linker. Symbols defined in an OFFSET table are kept internally, but no object is produced.

```
        OFFSET  *-START+$500
TSM1    DS.L    1
```

<Expression> is the value at which the offset table begins. The expression must be absolute and not contain forward, undefined, or external references.

If no <expression> is given, the last OFFSET address is used.

6.1.10.32 OPT - ASSEMBLER OPTIONS

Format: [<label>] OPT [<option>][,<option>]....

The OPT directive selects various assembly options during the assembly process. These are defined as follows:

```
        OPT     CRE,PDOS,NOLF
```

Default 0	Option 1	Bit	Description
NOOLD	OLD	24	Branch format (68020)
LF	NOLF	23	Output line feeds
?ON	?OFF	15	? Conditional assembly
|L	|R	14	| Conditional assembly
NOALT	ALT	13	Alter source
	PDOS	12	PDOS reserved symbols
NOCRE	CRE	10	Cross reference
WARN	NOWARN	9	Output warnings
NOBUG	BUG	8	List debug object
ARL	ARS	7	Absolute long/short
CEX	NOCEX	6	DC expansions
CL	NOCL	5	Conditional assembly list
FRL	FRS	4	Forward reference
MC	NOMC	3	Macro calls list
MEX	NOMEX	2	Macro expansion list
NOD	D{=m}	1	Dump symbol table
MB	NOMB	0	Print macro body
Tx			Assembly type
EMSK=#			Error mask
NLP=#			Number of lines/page
CID=#			Coprocessor ID (68881)
		30	Processor option selected
P=68010		29	68010 instructions enable (68010)
P=68020		28	68020 instructions enable (68020)
P=68881		27	68881 instructions enable (68881)
M68000	M68010		68000/68010 select

Note that Motorola assembler options MD and NOMD are included in MEX and NOMEX; and BRL and BRS are included in FRL and FRS.

If a label is included with the OPT directive, it is first loaded with the current options value and declared a 'SET' variable. This allows programs to capture the current option status, set new status, and then restore the old status when done.

6.1.10.32.1 ?ON/?OFF

? Conditional assembly. An assembly source line can be preceded by a question mark for line by line conditional assembly. The ?OFF option directs the assembler to change all lines beginning with a question mark to a comment line.

```
1 0/0000:7000      ? MOVEQ.L #0,D0
2        00008000    OPT     ?OFF
3 0/0002:          ? MOVEQ.L #1,D0
4 0/0002:            END
```

6.1.10.32.2 |L/|R

| Conditional assembly. Two different assembly instructions can be put on one line. The line must begin with a "|" character and the two instructions must be separated with another up-line (|) character. The |R option causes the instruction on the right to be assembled. The |L option or no option causes the instruction on the left to be assembled. In either case, the label field begins at the first character following the corresponding "|" character.

```
1 0/000000:000A     | DC.W 10 | DC.W -10
2          00004000  OPT  |R
3 0/000002:FFF6     | DC.W 10 | DC.W -10
4          00000000  OPT  |L
5 0/000004:000A     | DC.W 10 | DC.W -10
6 0/000006:          END
```

6.1.10.32.3 ALT/NOALT

Alter source. The ALT option directs the assembler to change the user-specified 68000 instructin opcode in order to optimize instructions whenever possible. This includes changing zero displacements to indirect register addressing and altering certain address registers and immediate instructions to avoid errors.

```
OPT ALT
```

Example:

```
 1i 0/00000000:4AFC                  CMP.w   (A1)+,(A2)+
**** ERROR 10 Illegal operand mode
 2i 0/00000006:4AFC                  EOR.w   #$FF,D0
**** ERROR 10 Illegal operand mode  [1/1]
 3i 0/00000010:4AFC4AFC              ADD.w   #8,A0
**** ERROR 10 Illegal operand mode  [1/3]
 4i 0/00000014:4AFC4AFC              ADD.w   #2,(A0)
**** ERROR 10 Illegal operand mode  [1/6]
 5i 0/0000001C:4AFC                  MOVE.w  D0,A7
**** ERROR 10 Illegal operand mode  [1/8]
 6  0/0000001E:31400000              MOVE.w  D0,0(A0)
 7                             *
 8                    00002000       OPT     ALT
 9  0/00000022:B549                  CMPm.w  (A1)+,(A2)+
10  0/00000028:0A4000FF              EORi.w  #$FF,D0
11  0/00000034:D1FC00000008          ADDa.l  #8,A0
12  0/0000003A:06500002              ADDi.w  #2,(A0)
13  0/00000042:3E40                  MOVEa.w D0,A7
14c 0/00000044:3080                  MOVE.w  D0,0(A0)
15  0/00000046:                      END
```

6.1.10.32.4 ARL/ARS

Absolute long/short. The ARS option directs the assembler to resolve all instructions using absolute addressing mode into the absolute short (16-bit) addressing mode whenever possible. The ARL option (the MASM default) causes all into absolute long addressing mode.

```
1 0/0000:13C000000300  MOVE.B D0,$300
2             00000080  OPT    ARS
3 0/0006:11C00300       MOVE.B D0,$300
4 0/000A:               END
```

6.1.10.32.5 BUG/NOBUG

List debug object. The BUG option directs the assembler to insert the actual object characters generated at the beginning of each source line in the list file.

```
           1      00000100    OPT   BUG
47008      2 0/0000:7008    L1 MOVEQ.L #8,D0
460FC      3 0/0002:60FC       BRA.S L1
4010240304 4 0/0004:01020304   DC.B  1,2,3,4
1000000000 5 0/0008:0/0000     END   LP1
```

6.1.10.32.6 CEX/NOCEX

DC expansions. The NOCEX option directs the assembler to expand only the first line of a defined constant directive in the list file.

```
1 0/0000:0000000100000002  DC.L 1,2,3,4,5,6
2        0000000300000004
3        0000000500000006
4                00000040  OPT  NOCEX
5 0/0018:0000000100000002  DC.L 1,2,3,4,5,6
6 0/0030:                  END
```

6.1.10.32.7 CID=#

The CID option sets the 68020 coprocessor identification field for F-line instructions. Default is 1.

```
1                      48000000 OPT      P=68881
2 0/00000000:F23C5000000A       FMOVE.W #10,FP0
3            0000F40048000000 OPT      CID=2
4 0/00000006:F43C5000000A       FMOVE.W #10,FP0
5 0/0000000C:                   END
```

6.1.10.32.8 CL/NOCL

Conditional assembly list. The NOCL option causes all unassembled source (because of conditional assembly) not to be listed in the list file.

```
 1 0/0000:          IFEQ 0
 2                 * ASSEMBLE THIS
 3                  ENDC
 4 0/0000:          IFNE 0
 5                 * DON'T ASSEMBLE
 6                  ENDC
 7        00000020  OPT NOCL
 8 0/0000:          IFEQ 0
 9                 * ASSEMBLE THIS
10                  ENDC
11                  ENDC
12 0/0000:          END
```

6.1.10.32.9 CRE/NOCRE

Cross reference. The CRE option directs the assembler to output a symbol cross reference to the 'LST=' file (if specified).

```
OPT CRE
```

6.1.10.32.10 D/NOD

Dump symbol table. The D option directs the assembler to dump all symbols to the object module as if they had been XDEFed. An optional mask can be included to selectively output symbols. The '*' character specifies a single wild card character while '@' specifies all match to end of symbol.

```
OPT D=L@
```

6.1.10.32.11 EMSK=#

Error mask. The EMSK option directs the assembler to ignore any errors with corresponding bits in the '#' number. Bits are numbered from left to right with the sign bit being 0.

```
1z 0/0000:4A           DC.B 10/0
*** ERROR 20 Division by zero
2  0000080000000000    OPT  EMSK=$00000800
3z 0/0001:0A           DC.B 10/0
4  0/0002:             END
```

6.1.10.32.12 FRL/FRS

Forward reference. The FRS option directs the assembler to resolve all unspecified forward references on first pass as short references.

```
1w 0/0000:60000006     BRA   L
*** WARNING 23 Branch could be short
2          00000010    OPT   FRS
3  0/0004:6002         BRA   L
4  0/0006:4E71         NOP
5  0/0008:           L END
```

6.1.10.32.13 LF/NOLF

Output line feeds. The NOLF option eliminates line feeds ($0A) from the list and object output files. This reduces the file sizes.

```
OPT NOLF
```

6.1.10.32.14 M68000/M68010

M68010 instructions. The M68010 option allows all 68010 instructions to be assembled.

```
1            20000000  OPT      M68010
2 0/00000000:42C0      MOVE.W   CCR,D0
3 0/00000002:4E7A8000  MOVEC.L  SFC,A0
4 0/00000006:4E7B1001  MOVEC.L  D1,DFC
5 0/0000000A:0E501800  MOVES.W  D1,(A0)
6 0/0000000E:4E74FFEC  RTD      #-4*5
7 0/00000012:          END
```

6.1.10.32.15 MB/NOMB

The MB/NOMB options select the listing of macro expansions. The MB option, where possible, will only list the macro header along with any code generated by the macro expansion.

```
1                        DATA MACRO
2                             DC&0    &1,&2
3                             ENDM
4                        *
5 0/00000000:          m      DATA.B 1,2,3
6 0/00000000:0102      a      DC.B    1,2
7            00000001         OPT     NOMB
8 0/00000002:0102      a      DATA.B 1,2,3
9 0/00000004:                 END
```

6.1.10.32.16 MC/NOMC

Macro calls list. The NOMC option causes the macro header not to be listed in the list file.

```
1                   M MACRO
2                     DC.B &1,&2
3                     ENDM
4                   *
5        00000008     OPT NOMC
6 0/0000:0102       a DC.B 1,2
7 0/0002:             END
```

6.1.10.32.17 MEX/NOMEX

Macro expansion list. The NOMEX option causes the expanded body of a macro not to be listed in the list file. Listing of individual lines of the macro expansion can be inhibited by inserting a minus sign as the first character of the line.

```
1                   M MACRO
2                     DC.B &1,&2
3                     ENDM
4                   *
5        00000004     OPT NOMEX
6 0/0000            m M 1,2
7 0/0002              END
```

6.1.10.32.18 NLP=#

Number of lines/page. The NLP option selects the number of lines per page in the list file before an automatic page throw is generated.

```
OPT NLP=30
```

6.1.10.32.19 OLD/NOOLD

The OLD/NOOLD options determine how the '.L' extension for branch instructions is handled. For 68000 and 68010, '.L' branch instructions are four bytes. For 68020, '.L' extensions are six bytes. The OLD option allows 68020 '.L' instructions to remain four bytes long. (It is advisable to use '.X' extensions to remove any confusion.)

```
1               50000000          OPT   P=68020
2               50000200          OPT   NOWARN
3   0/00000000:60FE         START BRA.B START
4   0/00000002:60FC               BRA.S START
5w  0/00000004:60FFFFFFFFFA       BRA.L START
6   0/0000000A:60F4               BRA.S START
7w  0/0000000C:60FFFFFFFFF2       BRA.X START
8               51000200          OPT   OLD
9   0/00000012:60EC               BRA.B START
10  0/00000014:60EA               BRA.S START
11w 0/00000016:6000FFE8           BRA.L START
12  0/0000001A:60E4               BRA.S START
13w 0/0000001C:60FFFFFFFFE2       BRA.X START
```

6.1.10.32.20 P=xxxxx

The P= option selects the type of Motorola processor for code generation. Currently the options are:

P=68010	68010 instructions enable (68010)
P=68020	68020 instructions enable (68020)
P=68881	68881 instructions enable (68881)

6.1.10.32.21 PDOS

PDOS reserved symbols. PDOS system constants are available by reference with the PDOS option. Only those symbols referenced are resolved at the beginning of the second pass.

```
1             00001000  OPT    PDOS
2 0/0000:1D7C0002044F   MOVE.B #2,PRT$(A6)
3 0/0006:               END
```

6.1.10.32.22 Tx

Assembly type. The Tx option inserts the character following the 'T' in the IDNT or O object record. Default is 'TA' for assembler. 'TP' is used in Pascal; 'TC' is used in C.

```
OPT TP
```

6.1.10.32.23 WARN/NOWARN

Output warnings. The NOWARN option disables any warning messages. However, the warning character will still appear at the beginning of the source line listing.

```
1a 0/0000:7080      LP1  MOVEQ.L #128,D0
**** WARNING 17 Arithmetic overflow
2w 0/0002:6000FFFC       BRA.L   LP1
**** WARNING 23 Branch could be short  [1/1]
3          00000200      OPT     NOWARN
4a 0/0006:7080      LP2  MOVEQ.L #128,D0
5w 0/0008:6000FFFC       BRA.L   LP2
6  0/000C:               END
```

6.1.10.33 ORG - ABSOLUTE ORIGIN

Format: [<label>] ORG[.qualifier] [<expression>] [<comments>]

The ORG directive changes the program counter to the value of the expression in the operand field. Subsequent statements are assigned absolute memory locations starting with the new program counter. <Expression> must be absolute and may not contain any forward, undefined, or external references.

```
        ORG     $200
ERROR   DS.L    1
```

If no operand <expression> is supplied, then the last absolute program counter is used.

The <qualifier> may be either an 'S' or 'L'. 'ORG.S' is interpreted as both 'ORG' and 'OPT FRS'. 'ORG.L' is interpreted as both 'ORG' and 'OPT FRL'.

6.1.10.34 PAGE - TOP OF PAGE

Format: PAGE

The PAGE directive advances the paper to the top of the next page. It does not appear in the program listing. No label or operand is allowed, and no machine code results.

```
        PAGE
```

6.1.10.35 PRINT - PRINT TO CONSOLE

Format: PRINT {'<string>'},{{$}<exp>}...

The PRINT directive allows the output of both strings and expression values during the assembly process to the console. If the expression begins with a dollar sign ($), then the value is output in hexadecimal.

```
PRINT 'I = ',I

PRINT 'I = $',$I
```

6.1.10.36 REG - DEFINE REGISTER LIST

Format: [<label>] REG <register list> [<comments>]

The REG directive assigns a value to <label> that can be translated into the register list mask format used by the MOVEM instruction. The REG directive acts like a SET directive and the label may be reassigned later.

```
DRL     REG     D0-D2/A0/A4
```

6.1.10.37 RORG - RELOCATABLE PC ADJUST

Format: [<label>] RORG[.qualifier] [<expression>] [<comments>]

The RORG directive adjusts the program counter within the current section.

If the <expression> is not given, then the last relocatable section and section address is selected.

The <qualifier> may be either an 'S' or 'L'. 'RORG.S' is interpreted as both 'RORG' and 'OPT FRS'. 'RORG.L' is interpreted as both 'RORG' and 'OPT FRL'.

```
        XREF    VALUE
        DC.L    VALUE
        RORG    *-4     ;BACKUP
        DC.L    10      ;DEFAULT TO 10
```

6.1.10.38 SECTION - PROGRAM SECTION

Format: [<label>] SECTION[.S] <expression>

The section directive causes the program counter to be restored to the address following the last location allocated in the section indicated by <expression> (or to zero if used for the first time).

<Expression> must range from 0 to 15. By default, the assembler begins with section 0, address 0.

```
        SECTION 0
START   XPMC    MES01   ;OUTPUT START
```

6.1.10.39 SET - REDEFINE ASSEMBLY CONSTANT

Format: [<label>] SET <expression> [<comments>]

The SET directive temporarily assigns the value of the expression in the operand field to the symbol in the label field. This symbol may be reassigned many times. The <label> is optional. A well-defined expression is not required on the first pass.

```
RL      SET     6*4
MMRL    REG     D0-D4/A5
```

6.1.10.40 SPC - SPACE BETWEEN SOURCE LINES

Format: SPC <expression>

The SPC directive outputs <expression> blank lines to the assembly listing.

```
SPC     10
```

6.1.10.41 TTL - TITLE

Format: TTL <title string>

The TTL directive uses the string argument as the heading for each page thereafter in the list file.

```
TTL     MASM ASSEMBLER 10/27/86
```

6.1.10.42 XDEF - EXTERNAL SYMBOL DEFINITION

Format: [<label>] XDEF <symbol>[,<symbol>...].

The XDEF directive outputs to the object file symbols and addresses to be used by the linker. For another way to externally define a symbol, see the EXTN directive.

```
XDEF    K$MASK
```

6.1.10.43 XREF - EXTERNAL SYMBOL REFERENCE

Format: [<label>] XREF[.S] [<section>:]<symbol>[,....

The XREF directive specifies symbols referenced in the current module but defined in other modules. If a <section> is specified, then only that section and symbol will resolve the reference. Otherwise, any matching symbol from any section will resolve the reference.

```
XREF    B$MAP
XREF.S  ERROR
```

The '.S' qualifier indicates that the XREF symbols will be linked into low address memory so that direct addressing of these symbols may be accomplished through absolute short mode. For another way to reference an external symbol, see the EXTN directive.

6.1.11 ASSEMBLER RESERVED WORDS

The MASM assembler has various types of reserved words. These include 68000 instructions, such as the 68010 or 68020 instruction set and the PDOS assembly primitives. Other reserved words include predefined assembler symbols.

6.1.11.1 ASSEMBLER 68000 OPERATORS

ABCD	ADD	ADDA	ADDI	ADDQ	
ADDX	AND	ANDI	ASL	ASR	
BCC	BCHG	BCLR	BCS	BEQ	
BGE	BGT	BHI	BHS	BLE	
BLO	BLS	BLT	BMI	BNE	
BPL	BRA	BSET	BSR	BTST	
BVC	BVS	CHK	CLR	CMP	
CMPA	CMPI	CMPM	DBCC	DBCS	
DBEQ	DBF	DBGE	DBGT	DBHI	
DBHS	DBLE	DBLO	DBLS	DBLT	
DBMI	DBNE	DBPL	DBRA	DBT	
DBVC	DBVS	DIVS	DIVU	EOR	
EORI	EXG	EXT	JMP	JSR	
LEA	LINK	LSL	LSR	MOVE	
MOVEA	MOVEM	MOVEP	MOVEQ	MULS	
MULU	NBCD	NEG	NEGX	NOP	
NOT	OR	ORI	PEA	RESET	
ROL	ROR	ROXL	ROXR	RTE	
RTR	RTS	SBCD	SCC	SCS	
SEQ	SF	SGE	SGT	SHI	
SHS	SLE	SLO	SLS	SLT	
SMI	SNE	SPL	ST	STOP	
SUB	SUBA	SUBI	SUBQ	SUBX	
SVC	SVS	SWAP	TAS	TRAP	
TRAPV	TST	UNLK			
MOVEC	MOVES	RTD			M68010 instructions

6.1.11.2 PDOS PRIMITIVES

The PDOS primitives described in chapter 4 of this manual are legal opcodes for the MASM assembler and may appear in the operation field

6.1.11.3 OPT PDOS WORDS

The following symbols are added to the MASM symbol table as referenced with the 'OPT PDOS' directive. Those ending in a dollar sign ($) are displacements into the Task Control Block (A6).

OPT PDOS

xxxx$(A6)

ACI$	EUM$	MWB$	TL1$	U1P$
BUM$	EXT$	PRT$	TL2$	U2P$
CHK$	FEC$	PSC$	TL3$	U4P$
CLB$	FLG$	SDK$	TRC$	U8P$
CLP$	FPA$	SDS$	TRP$	UNT$
CMD$	FPE$	SFI$	TRV$	ZDV$
CNT$	IMP$	SLV$	TSP$	
CSC$	KIL$	SPU$	TWO$	
EAD$	LEN$	TBE$	TW1$	
ECF$	MMF$	TID$	TW2$	
ERR$	MPB$	TLO$	TW3$	

Those ending in a period (.) are displacements into the system RAM (A5).

xxxx.(A5)

BCLK.	EVTM.	RL1.	TLTP.
BFLG.	EVTO.	RL2.	TPRY.
BIOS.	EVTS.	RL3.	TSKN.
BRKF.	F8BT.	RWCL.	UART.
CCNT.	FCNT.	SDAY.	URAT.
CHIN.	IORD.	SHRS.	USIM.
CHOT.	MAIL.	SMIN.	UTCB.
DFLG.	MAPS.	SMON.	UTIM.
E122.	OPIP.	SPTN.	UTYP.
E123.	PATB.	SSEC.	WADR.
E124.	PNOD.	SUIM.	WIND.
E125.	RDKA.	SYRS.	
EVTB.	RDKN.	TICS.	
EVTI.	RDKS.	TLCK.	

6.2 MASM20 68020 ASSEMBLER

MASM20 is a PDOS Motorola 68000/10/20 assembler. It accepts 68000 assembly mnemonics and directives, and outputs PDOS tagged or system object code. In addition, it supports the new 68020 addressing modes, new instructions, and 68881 floating point co-processor instructions.

Two-pass assembler

The assembler is a two-pass assembler. The first pass resolves all symbols. The second pass generates the object, listing, errors, and cross reference, if selected.

MASM20 is identical to MASM with the following additions:

1. Additional error messages.
2. Additional OPTions.
3. 68020 addressing modes.
4. Additional 68020 instructions.
5. Symbol and instruction extensions.
6. 68881 co-processor support.
7. Additional macro functions.

6.2.1 Additional error messages.

ERROR 340 = 68020 instruction or address mode
ERROR 341 = Illegal IS/I memory indirection
ERROR 342 = Expecting closing parentheses
ERROR 343 = Expecting comma
ERROR 344 = Illegal scale factor
ERROR 345 = Illegal {offset:width} format
ERROR 346 = Illegal register specification

6.2.2 Additional OPTions.

OLD	NOLD	68000 branch extensions
MB	NOMB	Print macro body
CID=#		Co-processor ID
P=68010		68010 Processor
P=68020		68020 Processor
P=68881		68881 Co-processor

6.2.3 68020 addressing modes.

```
 MODE 6 = d(An,Ri)
          ([bd,An,Ri{*scl}],od)
          ([bd,An],Ri{*scl},od)
          (bd,An,Ri{*scl})
MODE 73 = d(PC,Xi)
          ([bd,PC,Ri{*scl}],od)
          ([bd,PC],Ri{*scl},od)
          (bd,PC,Ri{*scl})
          MSP
          ISP
          VBR
          SFC
          DFC
          CACR
          CAAR
          FPn
```

6.2.4 Additional 68020 instructions.

Bcc	Supports 32-bit displacement
BFxxxx	Bit Field Instructions
BKPT	New Instruction Functionality
CALLM	New Instruction
CAS	New Instruction
CAS2	New Instruction
CHK	Supports 32-bit Operands
CHK2	New Instruction
CMP2	New Instruction
DIVS/DIVU	Supports 32-bit and 64-bit Operands
DIVSL/DIVUL	New Instruction
EXTB	New Instruction
ILLEGAL	New Instruction
LINK.L	Supports 32-bit displacements
MOVEC	Supports new control registers
MULS/MULU	Supports 32-bit operands
PACK	New Instruction
RTM	New Instruction
TRAPcc	New Instruction
TST	PC relative addressing
UNPK	New Instruction

6.2.5 Symbol and instruction extensions.

(<exp>).W and (<exp>).L supported.

New extensions of .X, .D, and .P.

6.2.6 68881 co-processor support.

Co-processor default set to 1.

FABS	FMOVECR
FACOS	FMOVEM
FADD	FMUL
FASIN	FNEG
FATAN	FNOP
FATANH	FREM
FBcc	FRESTORE
FCMP	FScc
FCOS	FSAVE
FCOSH	FSCALE.
FDBcc	FSGLDIV
FDIV	FSGLMUL
FETOX	FSIN
FETOXM1	FSINCOS
FGETEXP	FSINH
FGETMAN	FSQRT
FINT	FSUB
FINTRZ	FTAN
FLOG10	FTANH
FLOG2	FTENTOX
FLOGN	FTRAPcc
FLOGNP1	FTST
FMOD	FTWOTOX
FMOVE	

DC/DS Floating point Constants

6.2.7 Additional macro functions.

New MACRO parameters: &*, &@, and &{s,e}n.

&& = &
&@ = NUMBER OF ARGUMENTS
&# = MCT_(A6)
&* = MACRO HEADER (MSV_)
&0 = MACRO EXTENSION (MEX_)
&1-&9 = PARAMETER
&{s,e}n = PARAMETER EXTRACT
&<symbol>& = SYMBOL VALUE

6.3 QLINK

The QLINK linker is a single-pass, in-memory linker which accepts PDOS tagged object and outputs S-records, PDOS system files, or PDOS tagged object.

QLINK - single-pass memory linker

The following are the syntax definitions used in describing the linker commands:

< > = string argument
[ ] = number (hex must be preceded by $)
{ } = optional argument

All hexadecimal numeric inputs require a dollar sign ($) to precede the number.

Section high and low addresses are available as parameters to linker commands. The low addresses are named 'Q$L0' through 'Q$LF' for section 0 through section 15 respectively. Likewise, 'Q$H0' through 'Q$HF' are replaced by section 0 through section 15 high addresses respectively. These high section addresses are equal to the address that would be loaded next with data in that section. In other words, Q$H0 is equal to the last address of section zero that was loaded plus one.

Q$L0 - Q$LF

OUTPUT Q$L0,Q$H0

The QLINK linker supports arithmetic statements in all command line expressions. This includes the standard assembler operators plus QLINK defined symbols.

6.3.1 QLINK COMMANDS

Following is an explanation of the commands available with the QLINK linker.

6.3.1.1 ADD

Definition: Write data to QLINK buffer
Aliases: AD
Format: ADD{.W or L} [sec]{..,[value]}

The ADD command writes either a word or long word of data to a QLINK section.

```
ADD.W 0,$A038    ;insert XBUG
```

6.3.1.2 ALIAS

Definition: Equate symbols
Aliases: A, AL
Format: ALIAS <symbol1>,<symbol2>

The ALIAS command equates two symbols to the same value. <symbol1> is equated to <symbol2>.

```
ALIAS SYRAM,SYSRAM
```

6.3.1.3 BASE

Definition: Set memory buffer base
Aliases: B, BA
Format: BASE [addr]

The BASE command sets the memory buffer base address. This allows you to window into the target logical address space. This is essential since your task memory space may not be large enough to buffer the complete object or more commonly, the link addresses are way beyond the end of your buffer (such as object for addresses $FFFFFF00 to $FFFFFFFF).

```
BASE $F0000
```

6.3.1.4 BITMAP

Definition: Create relocation bitmap
Aliases: BI, BIT
Format: BITMAP <S or E>

The BITMAP command outputs relocation code and bitmap to section zero to transform positon dependent to position independent code. The S or start option is first executed to output relocation code. After the load process, the E option must be executed to output a bitmap for relocating long words within the program.

(6.3.1.4 BITMAP continued)

The relocation code is as follows:

```
*************************************************
*       RELOCATION CODE
*************************************************
*
        XREF.1  S$OSZE
        SECTION 0
S$START IDNT    1.0
*
BSTART  MOVEA.L #S$OSZE,A0      ;GET CODE SIZE
        LEA.L   BSTART(PC),A1   ;PROGRAM START
        MOVE.W  #$6026,(A1)     ;MODIFY ENTRY
        ADDA.L  A1,A0           ;BIT MAP PTR
        MOVE.L  A1,D3           ;SAVE BASE
        MOVE.L  (A0)+,D2        ;BIT MAP WORDS
*
@0002   MOVE.L  (A0)+,D0        ;WORD OF BITMAP
        MOVEQ.L #32-1,D1        ;32 BITS/WORD
*
@0004   LSL.L   #1,D0           ;RELOCATE?
          BCC.S @0006           ;N
        ADD.L   D3,(A1)         ;Y, ADD BASE
*
@0006   ADDQ.W  #2,A1           ;NEXT
          DBF   D1,@0004        ;INNER LOOP
        SUBQ.L  #1,D2           ;DONE?
          BNE.S @0002           ;N
*
*       BEGIN PROGRAM EXECUTION
```

The end relocation bit map is defined as follows:

```
*************************************************
*       BITMAP CODE
*************************************************
*
BITMAP  DC.B 'OAS$BITMAP   1  00702861456'
        DC.B 'C 6S$OSZE'
BM1     DC.B '000000005'        ;SECTION 0 SIZE
BM2     DC.B '0000000020'       ;# OF BIT MAP LONG WORDS
BM3     DC.B '00000000'         ;DEFINE BIT MAP STORAGE
        DC.B '1000000000'       ;END
        DC.B ':',0
*
*************************************************
*       END BITMAP CODE
*************************************************
```

6.3.1.5 COMMAND

Definition: Execute command file
Aliases: C, CO, COM
Format: COMMAND <filename>

The COMMAND command executes a procedure file from the linker.

```
COMMAND LINK2
```

6.3.1.6 DEFINE

Definition: Define symbol
Aliases: D, DE, DEF
Format: DEFINE <symbol>,[value]

The DEFINE command defines a symbol in the linker dictionary. The symbol will be absolute unless preceded by a decimal section number and colon.

```
DEFINE SYRAM,$9000
```

6.3.1.7 DISK

Definition: Load disk image
Aliases: DI
Format: DISK [dsk]

The DISK command loads the PDOS disk image specified by [dsk] into section 0. The number of sectors loaded is the same as the disk size.

6.3.1.8 DUMP

Definition: Dump buffer memory
Aliases: DU
Format: DUMP {[adr1],[adr2]}

The DUMP command displays a hexadecimal and ASCII memory dump from the link buffer to your console.

```
DUMP $1000,$2000
```

6.3.1.9 END

Definition: Finish link & output results
Aliases: E, EN
Format: END {[address] or <symbol>}

The END command causes the linker to finally output the linked object in the object type selected. The start address is optionally specified by an address number or a linker symbol.

```
END
END $1000
END .START
```

6.3.1.10 EVEN

Definition: Put section data on even word boundary
Aliases: EV
Format: EVEN {<section>,<mask>}

The EVEN command causes the linker to put the current highest address and all subsequent section addresses on an even word boundary.

```
IN FILE1
EVEN
```

Selective sections can be put on powers of two boundaries by including a section parameter followed by a mask. The mask is added to the highest loaded address and then the address is masked with the 1's complement of the mask.

6.3.1.11 GROUP

Definition: Group contiguous sections
Aliases: G, GR
Format: GROUP [gsec]{..,[sec]}

The GROUP command groups together two or more sections as if they were from the same section. The first section [gsec] is the base section and other following sections are changed to that section number.

```
GROUP 0,1,2
```

6.3.1.12 HELP

Definition: List QLINK commands
Aliases: H, HE
Format: HELP

The HELP command outputs the linker commands and their parameter formats to your console. HELP

```
*HE
ADD{.W or L} [sec]{..,[value]}  Add to section
ALIAS <symbol1>,<symbol2>       Equate symbols
BASE [addr]                     Set memory buffer base
BITMAP <BEGIN or END>           Create relocation bitmap
COMMAND <filename>              Command file
DEFINE <symbol>,[value]         Define symbol
DUMP {[adr1],[adr2]}            Dump buffer memory
END {[address] or <symbol>}     Finish link
EVEN {<section>,<mask>}         Even module boundaries
GROUP [gsec]{..,[sec]}          Group contiguous sections
IGNORE [sec]{..,[sec]}          Ignore section data
INPUT <filename>                Link file
LIBRARY <filename>              Link library
MAP <options>{,<filename>}      Output link map
OBJECT {[sadr],[eadr]}          Output OB object
OFFSET [section],[addr]         Set section offset PC
OUTPUT <filename>               Link object output file
PARTIAL {<section list>{,XDEF}} Output partial link
QUIT                            Exit linker

Strike any key

RELINK <section>,<base>         Relink to new base
RENAME <sym1>,<sym2>            Rename symbol
RESTART                         Reset linker
SECTION [section],[addr]        Set section address
SRECORD {[sadr],[eadr]{,[adr]}} Output S-records
SYFILE {[sadr],[eadr]}          Output SY object
UNDEFINED {<filename>}          List unresolved symbols
WRITE [dsk],[sec],[ad1],[ad2]   Write memory image
XDEF <symbol>                   Partial external define
ZERO                            Zero buffer

MAP options: A=Aliases       M=Multiply defined
             B=Memory base   O=Overflows
             D=Symbols       R=References
             F=File list     S=Sections
             G=Groups        U=Undefined
             I=Ignores       V=Resolved
```

6.3.1.13 IGNORE

Definition: Ignore section data
Aliases: IG
Format: IGNORE [sec]{..,[sec]}

The IGNORE command tells the linker to define all symbols in the ignored sections but not to store or load any object from those sections.

IGNORE 1,2,3

6.3.1.14 INPUT

Definition: Input file to linker
Aliases: I, IN
Format: INPUT <filename>

The INPUT command loads a PDOS tagged object file into the memory buffer. All symbols are defined and resolved if possible. For each file input, a new entry in the file link map is made.

INPUT FILE1:OBJ

6.3.1.15 LIBRARY

Definition: Input link library
Aliases: L, LI, LIB
Format: LIBRARY <filename>

The LIBRARY command loads files from a library file generated by the MLIB or MLIBGEN programs. A file is loaded when it contains an unresolved XDEF entry. For each file loaded, the library file is rewound and the process repeated. The command terminates after it has made a pass and no file was loaded.

LIBRARY PASCAL:LIB

6.3.1.16 MAP

Definition: Output link map
Aliases: M, MA
Format: MAP <options>{,<filename>}

The MAP command outputs the current linker symbols, definitions, addresses, and other information. The <options> parameter selects the information to be displayed. These options are defined as follows:

MAP ALL
MAP UFRS

A = Aliases
B = Base
D = Definitions
F = Files
G = Groups
I = Ignored sections
M = Multiply defined
O = Resolving overflows
R = References
S = Sections
U = Undefined references
V = Resolved reference values

If no <option> is given, the default is 'FGOSU'. The 'ALL' option will output all data.

6.3.1.17 OBJECT

Definition: Output OB object
Aliases: OB, OBJ
Format: OBJECT {[sadr],[eadr]}

The OBJECT command sets the linker to output PDOS tagged object when the END command is executed. It optionally sets the start [sadr] and end [eadr] addresses of the buffer that will be output. These addresses are the actual QLINK resolved addresses and NOT buffer or section offsets. The last byte output will be from address [eadr]-1.

OBJECT $1000,$9000

6.3.1.18 OFFSET

Definition: Set section offset PC address
Aliases: OF, OFF
Format: OFFSET [section],[addr]

The OFFSET command sets a new section PC address.

OFFSET 0,$9000

6.3.1.19 OUTPUT

Definition: Select link output file
Aliases: O, OU, OUT
Format: OUTPUT <filename>

The OUTPUT command selects the output file for the linked object.

```
OUTPUT #OBJECT
```

6.3.1.20 PARTIAL

Definition: Output partial link
Aliases: P, PA, PAR
Format: PARTIAL {<section list>{,XDEF}}

The PARTIAL command outputs all code from the specified <section list> to the object file along with external definitions and resolving information for unresolved references. If the ',XDEF' parameter follows the section list, then only the associated externally defined symbols are output.

```
PARTIAL 0-2/6

PARTIAL 0,XDEF
```

The format of the section list calls for individual sections separated by a slash (/) and/or consecutive sections reduced to the start and end section separated by a minus sign (-).

See also the XDEF command, section 6.3.1.30.

6.3.1.21 QUIT

Definition: Exit linker
Aliases: Q, QU
Format: QUIT

The QUIT command exits from the linker back to PDOS.

```
QUIT
```

6.3.1.22 RELINK

Definition: Relink section addresses to new base
Aliases: REL
Format: RELINK [section],[addr]

The RELINK command unresolves all resolving equations involving a section [section] and then again resolves the variables to the new base address [addr]. This allows the moving of a section dynamically during the link process. It is used mainly by the C compiler to locate the data section at the end of the code section.

```
RELINK 2,Q$HO
```

See also IGNORED, section 6.3.1.13.

6.3.1.23 RENAME

Definition: Rename linker symbol
Aliases: RE, REN
Format: RENAME <symbol1>,<symbol2>

The RENAME command renames a linker symbol. The new symbol name would apply for all subsequent inputs. The old symbol is subsequently undefined.

```
RENAME .MAIN,PRGM1
```

6.3.1.24 RESTART

Definition: Reset linker
Aliases: R, RE, RES
Format: RESTART

The RESTART command restarts the linker, resets all addresses, clears any grouping, ignores, and section bases, and closes all open files.

```
RESTART
```

6.3.1.25 SECTION

Definition: Set section base address
Aliases: S, SE, SEC
Format: SECTION (section),[addr]

The SECTION command sets the absolute base [addr] for a (section).

SECTION 0,$800

6.3.1.26 SRECORD

Definition: Output S-record object
Aliases: SR, SREC
Format: SRECORD {[sadr],[eadr]{,[adr]}}

The SRECORD command selects the S-record format for output. The object is output when the 'END' command is executed. It optionally sets the start [sadr] and end [eadr] addresses of the buffer that will be output. These addresses are the actual QLINK resolved addresses and NOT buffer or section offsets. The last byte output will be from address [eadr]-1.

SRECORD $1000,$9000,$F00000

6.3.1.27 SYFILE

Definition: Output SY object
Aliases: SY, SYF
Format: SYFILE {[sadr],[eadr]}

The SYFILE command selects the PDOS system format for output. The object should be position independent but QLINK does not check this first. The object is output when the 'END' command is executed. It optionally sets the start [sadr] and end [eadr] addresses of the buffer that will be output. These addresses are the actual QLINK resolved addresses and NOT buffer or section offsets. The last byte output will be from address [eadr]-1.

SYFILE $1000,$9000

6.3.1.28 UNDEFINED

Definition: List unresolved symbols
Aliases: U, UN, UND
Format: UNDEFINED {<filename>}

The UNDEFINED command outputs any unresolved references and any undefined symbols to your console.

```
UNDEFINED #UNDLIST
```

6.3.1.29 WRITE

Definition: Write memory image to disk
Aliases: W, WR
Format: WRITE {[dsk],[sec],[ad1],[ad2]}

The WRITE command uses the read/write primitives of PDOS to output a memory image for the link buffer to a disk.

```
*WRITE 0,2368,$800,$9800
VERIFY (Y/N)?Y
*WRITE
      Disk=0
    Sector=2368
Start addr=$800
  End addr=$9800
VERIFY (Y/N)?Y
```

6.3.1.30 XDEF

Definition: Define symbol external
Aliases: X, XD
Format: XDEF <symbol>

The XDEF command declares a linker symbol to be externally defined. When the XDEF command is used in conjunction with the PARTIAL command, only the specified symbols are written to the object file. Without the XDEF command, PARTIAL saves all external symbols in the output file.

XDEF also causes PARTIAL to output all symbols and code as section zero.

6.3.1.31 ZERO

Definition: Zero buffer
Aliases: Z, ZE
Format: ZERO

The ZERO command zeros the linker buffer and resets the linker.

```
ZERO
```

6.3.2 QLINK ERROR DEFINITIONS:

ERROR #501 = ILLEGAL COMMAND
ERROR #502 = ILLEGAL NUMBER
ERROR #503 = ILLEGAL SECTION SPECIFICATION
ERROR #504 = ILLEGAL SYMBOL
ERROR #505 = TOO MANY COMMAND FILES
ERROR #506 = PDOS CLOSE ERROR
ERROR #507 = PDOS OPEN ERROR
ERROR #508 = PDOS LOAD ERROR
ERROR #509 = 'OB' or 'SY' FILE REQUIRED
ERROR #510 = MEMORY SIZE EXCEEDED
ERROR #511 = ILLEGAL OBJECT TAG
ERROR #512 = INVALID ADDRESS RANGE
ERROR #513 = PDOS READ ERR
ERROR #514 = ILLEGAL OPTION
ERROR #515 = ARITHMETIC OVERFLOW
ERROR #516 = DIVISION BY ZERO
ERROR #517 = PDOS WRITE ERROR
ERROR #518 = ILLEGAL SECTION GROUPING
ERROR #519 = NESTING ERROR
ERROR #520 = FIELD OVERFLOW
ERROR #521 = SYMBOL NOT FOUND
ERROR #522 = SYMBOL ALREADY DEFINED
ERROR #523 = UNDEFINED SYMBOL
ERROR #524 = MEMORY OVERFLOW

6.3.3 LINKER EXAMPLE

```
x>QLINK                                                     Execute PDOS linker
PDOS 68k Quick Linker
ERII, Copyright 1983-86
*ZERO                                                       Zero load area
*SECTION 0,0                                                Set Section 0 at address $00000000
*GROUP 0,1                                                  Group Section 1 into Section 0
*INPUT PMAIN:OBJ                                            Input linking modules
ENTRY ADDRESS=00000000
*INPUT T:POB
*LIBRARY LIB1:LIB                                           Input libraries.  Only modules
*LIBRARY LIB2:LIB                                           XREFed are loaded.
 INPUT MPEND:OBJ                                            {Module names list as loaded}
 INPUT MPUNLNK:OBJ
 INPUT MPERROR:OBJ
 INPUT MPLINK:OBJ
 INPUT MPRDLNF:OBJ
 INPUT MPGETCH:OBJ
 INPUT MPIOOK:OBJ
 INPUT MPWRLNF:OBJ
 INPUT MPPUTCH:OBJ
 INPUT MPWRSTF:OBJ
 INPUT MPFPUTL:OBJ
*MAP SFU                                                    Look at map options Sections,
                                                              Files, and Undefined
INPUT FILE MAP:
INDEX FILE NAME      TYP IDNT        R  V   DATE     TIME   SECTION ADDRESSES
  1    PMAIN:OBJ      A  M.AIN       1  0  16-Sep-83 14:15  0/00000000 0000008B
  2    T:POB          P  PTEMP:PSR   1  0  17-Oct-83 10:02  0/0000008C 0000012D
  3    MPEND:OBJ      A  P.END       1  0  16-Sep-83 15:39  0/0000012E 00000181
  4    MPUNLNK:OBJ    A  U.NLNK      1  0  16-Sep-83 14:28  0/00000182 00000187
  5    MPERROR:OBJ    A  P.ERROR     1  0  21-Sep-83 11:43  0/00000188 000001E9
  6    MPLINK:OBJ     A  L.INK       1  0  16-Sep-83 14:14  0/000001EA 000001F7
  7    MPRDLNF:OBJ    A  R.DLNF      1  0  16-Sep-83 11:58  0/000001F8 00000233
  8    MPGETCH:OBJ    A  G.ETCH      1  0  21-Sep-83 11:58  0/00000234 00000323
  9    MPIOOK:OBJ     A  I.OOK       1  0  16-Sep-83 14:13  0/00000324 0000034F
  10   MPWRLNF:OBJ    A  W.RLNF      1  0  16-Sep-83 14:21  0/00000350 00000381
  11   MPPUTCH:OBJ    A  P.UTCH      1  0  16-Sep-83 14:16  0/00000382 000003BF
  12   MPWRSTF:OBJ    A  W.RSTF      1  0  16-Sep-83 14:22  0/000003C0 00000425
  13   MPFPUTL:OBJ    A  F.PUTL      1  0  16-Sep-83 14:22  0/00000426 0000059F

SECTION       BASE        LOWEST      HIGHEST
   0        00000000     00000000    000005A0

UNRESOLVED EXTERNAL DEFINITIONS: NONE

UNRESOLVED EXTERNAL REFERENCES: NONE

*MAP ALL,#MAP                                               Output all map options to file 'MAP'
*SYFILE                                                     Generate SY file
*OUTPUT #T                                                  Specify output file
*END                                                        Linker outputs now
*QUIT                                                       Exit linker
x>RC
```

CHAPTER 7

PDOS UTILITIES

This chapter describes the user and system-specific utilities of PDOS, along with the abort and virtual port facilities. A PDOS utility is an auxiliary program that resides on the disk. It is invoked by specifying the name of the utility along with any desired command line parameters. If no command line parameters are given and the utility requires a parameter, it will prompt for the information it needs. PDOS facility files are not user programs, but files which are run as background tasks. Refer to their appropriate sections for proper usage.

PDOS utility programs are distributed on the UTILITY disk of the distribution set in both executable and source file format. The knowledgeable user can thus refer to the code to gain additional understanding of how the utility works. You may customize any of these utilities to suit your individual needs; however, your modifications are not supported by Eyring.

USER UTILITIES

Continued on next page . . .

(TABLE OF CONTENTS cont.)

SYSTEM FACILITIES

SYSTEM-SPECIFIC UTILITIES

Other system-specific utilities are described in the Installation and Systems Management guide for your system.

The following utilities are not covered in this chapter:

MASM - Assembler	Covered in detail in chapter 6
QLINK - Quick Linker	" "
MEDIT - Screen Editor	Covered in detail in chapter 5
MEDITCON - Editor Configurator	" "

7.1 MBACK - DISK BACKUP

Name: MBACK
Function: Back up or copy disk
Format: >MBACK
>MBACK <source disk>,<dest. disk>,<# of sectors>,<Y>
>MBACK <source disk{/start sector}>,<dest disk{/start sector}>,<{# of sectors or F}>{,Y}

Restrictions:

This utility overwrites 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. To back up your disks using MBACK, consult the PDOS Installation and Systems Management guide for your hardware system.

In the following discussion, "disk number" refers to the PDOS disk unit or device number. For example, disk numbers 0-1 usually refer to floppy drives, disk number 8 is generally a RAM disk, and other disk numbers are typically hard disk paritions. If 100 is added to the floppy disk number, it means the unbiased disk rather than the logical disk unit. A PDOS logical floppy disk usually skips track 0, leaving that space for system-specific boot information. In the following examples, if the disk number is greater than 100, MBACK will copy the floppy disk starting with track 0. Otherwise, it will copy the floppy starting with track 1 (excluding the manufacturer's track).

Description:

The MBACK utility performs a sector-by-sector disk copy using one or two disk drives. MBACK 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 source and destination. 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.

The 'start sector' parameter permits you to back up an image of a large hard disk onto floppy-sized disks. You can thus restore the hard disk image from the floppy disks.

Continued on next page...

(7.1 MBACK - DISK BACKUP continued)

Next, the program prompts for the number of sectors to be copied. This number varies from system to system. For 5 1/4" double density, double sided 96 TPI floppies, the total number of sectors is 2560 (or 2552 if using a Motorola disk card). Since PDOS reserves track 0 for hardware-specific information, only 2560-32 sectors are usable for a PDOS data disk. If there is a PDOS boot on a floppy, only 2336 sectors are available for data storage. Thus, to back up a 5 1/4" floppy (let's assume drive 0), you need to know if you are backing up the entire disk, including hardware specific track 0 (use 2560 on disk 100), or just the PDOS portion of a disk including boot (use 2528 on disk 0), just the data without the boot (use 2336 on disk 0) or a data disk with no boot (use 2528 on disk 0). Before MBACK prompts for this information it reads the directory of the source disk and calculates a default number of sectors. Generally, this number is correct and you can simply type a [CR] to use it.

MBACK then prints a "READY" prompt. If you have not already done so, insert the original disk in the specified source drive and, if using two drives, insert the target disk in the destination drive. When you are ready, type 'Y'.

MBACK reads and displays the name of the source disk disk so that you can verify the transfer. (If you are backing up the disk including track 0, the name displayed will not make sense. This is normal.) When you enter a 'Y' to this last question, the disk duplication begins. As many sectors as possible are read into the task's memory from the source disk and then written to the destination 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.

WARNING! MBACK should only be used to copy to a disk of the same size as the original unless the image backup mode is being used. If you should, for instance, copy a floppy disk onto a large hard disk unit, the hard disk unit will assume the directory and storage capacity of a floppy disk. The best way to move files from floppy to hard disk is to backup (using MBACK) to a floppy-sized hard disk unit and then copy file by file from the floppy-sized hard disk unit to the large hard disk unit. The MTRANS utility or TF monitor command are useful for file-by-file transfers.

Continued on next page...

(7.1 MBACK - DISK BACKUP continued)

Examples:

Copy all tracks (including 0) of a 5 1/4" floppy disk using only drive 0. (When using track 0, the disk name may be changed or destroyed.)

```
x>MBACK
68K PDOS Disk Backup Utility
  Source: (Disk# or Disk/Sector) = 100
  Destination: (Disk# or Disk/Sector) = 100
  Insert source disk in drive 100.  Hit <CR>
  Number of sectors (# or 'F') = 2560
  Ready?Y
  Backup '...............'?Y
  Insert source disk in drive 100.  Hit <CR>....
Reading sector 0..2483
  Insert destination disk in drive 100.  Hit <CR>....
Writing sector 0..2483
  Insert source disk in drive 100.  Hit <CR>....
Reading sector 2496..2559
  Insert destination disk in drive 100.  Hit <CR>....
Writing sector 2496..2559
  SUCCESS!  Disk Name = ...............'
```

Back PDOS disk 13 (a floppy image) onto floppy disk 0 letting MBACK calculate the size of the transfer. Specify all necessary parameters on the command line.

```
x>MBACK 13,0,,
68K PDOS Disk Backup Utility
  Source: (Disk# or Disk/Sector) = 13
  Destination: (Disk# or Disk/Sector) = 0
  Number of sectors (# or 'F') = 2528
  Ready?Y
  Backup 'C COMPILER 1.0..'?Y

Reading sector 0..2483
Writing sector 0..2483
Reading sector 2496..2527
Writing sector 2496..2527
  SUCCESS!  Disk Name = C COMPILER 1.0..'
```

Continued on next page . . .

(7.1 MBACK - DISK BACKUP continued)

Back PDOS disk 2 (on Winchester) onto multiple floppy disks (PDOS disk 0).

```
x>MBACK 2,0,2500,Y
x>MBACK 2/2500,0,2500,Y
x>MBACK 2/2500,0,2500,Y
  . . .etc. until disk is transferred.
```

Restore PDOS disk 2 (on Winchester) from floppy disks (PDOS disk 0).

```
x>MBACK 0,2,2500,Y
x>MBACK 0,2/2500,2500,Y
x>MBACK 0,2/5000,2500,Y
  . . .etc. until disk is restored.
```

7.2 MCHATLE - CHANGE ATTRIBUTES/LEVEL

Name: MCHATLE
Function: Change attributes and levels of selected files
Format: >MCHATLE
>MCHATLE @:@;@/<disk #>,{<attribute>},{<level #>}

Restrictions: Cannot use level 255.

Description:

The MCHATLE utility changes the attributes and/or the directory level of a selected group of files to the specified value. The file descriptor string is the same as that used in MTRANS and MLDIR. 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, DR, 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 MCHATLE in the command line or by prompts from the utility.

Examples:

```
x>MCHATLE @:SR;@,TX
68K Change File Attributes
File mask = @:SR;@
     Type = TX
    Level =
PNETS:SR;1      TX
PSPELL:SR;1     TX
MPDOSK:SR;1                 MPDOSK:SR;1     TX
MPDOSB:SR;1     TX
```

Set all files with an 'SR' extension to have a file attribute of 'TX'.
Specify all options on the command line.

```
x>MCHATLE
68K Change File Attributes
File mask = @:SR;@
     Type = TX
    Level = 2
PNETS:SR;1      TX          PNETS:SR;2      TX
PSPELL:SR;1     TX          PSPELL:SR;2     TX
MPDOSK:SR;1     TX          MPDOSK:SR;2     TX
MPDOSB:SR;1     TX          MPDOSB:SR;2     TX
x>
```

Set all files with an 'SR' extension to have a file attribute of 'TX' and rename them to level 2.
Run the program and set the options interactively.

7.3 MDCOMP - DISK FILE COMPARE

Name: MDCOMP
Function: Compare disk files
Format: >MDCOMP
>MDCOMP <disk #1>,<disk #2>,<file mask>,<outfile>{/<options>}

Restrictions: To compare driver files, you must first change the file attributes. Restore the 'DR' attribute after the compare is made.

Description:

The MDCOMP utility compares multiple ASCII files from different disk units according to a file mask. The differences are noted in the output file along with a list of all files not compared. This utility is useful in documenting updates to source programs.

MDCOMP begins by building a directory list from each disk unit. It then compares files whose names match in the two lists. As many lines as possible are read from each file. Lines are compared until a difference is found at which time further searching looks for a match again. The utility prints the differences to a list file, or to the screen if no output file is specified.

Three parameters may be specified following the output filename. These are line length, sync length, and maximum difference block length. The line length parameter defines the maximum number of characters in a "line" for purposes of comparison. The default is 78 characters. The sync length defines how many lines in a row have to match in the two files before the data can be considered be considered equivalent. The default is that three (3) lines have to match for the two lines to be synchronized. The maximum difference block length defines the size of the largest difference to consider before aborting the comparison because the two files are hopelessly different. The default is 50 lines, meaning that if MDCOMP goes for 50 lines without synchronizing the files, it will stop comparing those two files.

Two blank lines follow after each difference block. After comparing each pair of files, MDCOMP prints the number of differences to the screen.

Continued on next page...

```
68K PDOS Disk File Compare Utility
  First Disk # = <disk #1>
 Second Disk # = <disk #2>
Directory Mask = <file mask>
        Output = <out filename> {/<options>}

Options:        /L=XX   Line length
                /S=XX   Sync Length
                /B=XX   Max difference block length
```

(7.3 MDCOMP - DISK FILE COMPARE continued)

All compare utility parameters can be specified from the command line. The file mask uses '@' for all match, '*' for wild card character, and '/xx' for file type specification. Control parameters follow the output file name and are delimited by a backslash.

Example:

```
       x>MDCOMP 7,10,@:SR;@,,
       68K PDOS Disk File Compare Utility 07/22/85
         First Disk # = 7
        Second Disk # = 10
       Directory Mask = @:SR;@
       Output =

FILE 1: EXIT:SR;99/7                 FILE 2: EXIT:SR;199/10
====================================  ====================================

****** BEGINNING OF FILE ******       ****** BEGINNING OF FILE ******
* EXIT:SR                             * THIS IS FOR STOPPING THE AC FILE AND
*                                     EXIT XRCN
EXIT XEXT                              RTS
 RTS                                   END
 END EXIT                             .....................................

1 Difference.

FILE 1: ARGS:SR;217/7                FILE 2: ARGS:SR;2/10
====================================  ====================================

*  19 Oct 84    converted to PDOS     *  19 Oct 84    converted to PDOS
*  15 Feb 85    changed mnemonics move *
*                                     .....................................

        ADDA.L  D2,A3                         ADDA.L  D2,A3
        MOVEA.L 4(A7),A2                      MOVEA.L  4(A7),A2
        ADDQ.L  #1,D3                         ADDQ.L  #1,D3

2 Differences.

FILE 1: DATE:SR;217/7                FILE 2: DATE:SR;2/10
====================================  ====================================

0 Differences.

Files Not Compared:

         RESUME:SR;99/7
         SUSP:SR;99/7
         ERRMSG:SR;217/7
         HELLO:SR;217/7
```

7.4 MDDMAP - DISK MAP

Name: MDDMAP
Function: Disk diagnostic -- read files by links
Format: >MDDMAP

Restrictions: None.

Description:

The MDDMAP 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 starting sector of a contiguous block of sectors is listed followed by the ending 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.

Continued on next page...

(7.4 MDDMAP - DISK MAP continued)

Example:

```
x>MDDMAP
68K PDOS Disk Diagnostic Mapper Utility
  Disk # = 20[CR]
  Output File Name =[CR]
*Disk Diagnostic Map

  Disk Name = WDISK #20
  Files = 78/160
  Boot sector = 0
  PDOS Sectors = 2368
  Disk Density = D

 10  B00        OB     2/2    16:37 25-Apr-84  16:37 25-Apr-84
     22-23

 10  B01        OB     2/2    16:37 25-Apr-84  16:37 25-Apr-84
     24-25

 2   B01:SR     TX     5/5    16:37 25-Apr-84  16:37 25-Apr-84
     26-30

 10  B02        OB     2/2    16:37 25-Apr-84  16:37 25-Apr-84
     31-32

 2   B02:SR     TX     5/5    16:37 25-Apr-84  16:37 25-Apr-84
     33-37

 10  B03        OB     2/2    16:37 25-Apr-84  16:37 25-Apr-84
     38-39

 2   B03:SR     TX     5/5    16:37 25-Apr-84  16:37 25-Apr-84
     40-44

 10  B04        OB     2/2    16:37 25-Apr-84  16:37 25-Apr-84
     45-46

 2   B04:SR     TX     5/5    16:37 25-Apr-84  16:37 25-Apr-84
     47-51

 10  B05        OB     2/2    16:37 25-Apr-84  16:37 25-Apr-84
     52-53

         .
         .
         .

Total Spoiled Files = 0
Total Bad Sectors = 0
x>_
```

7.5 MDDUMP - DISK DUMP

Name: MDDUMP
Function: Dump and alter disk sector
Format: >MDDUMP

Restrictions: Can re-write sectors external to PDOS.

Description:

The MDDUMP utility dumps sectors from a disk in hex and ASCII format to the screen. 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 a [CTRL-N], thus aborting the display of the whole sector. Use a [CTRL-D] to revert back to the start sector prompt. To temporarily stop the list, strike the space bar. Another space will start the list again. An [ESC] will exit MDDUMP.

^N select next sector
^D abort dump, prompt for new sector
[ESC] exit to PDOS monitor

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 [CTRL-C] will return to the main MDDUMP 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 MEDIT also work in alter mode: i.e., left=backspace (^H), right=formfeed (^L), up=vertical tab (^K), down=linefeed (^J). Change the buffer data by entering the new values in hex. To write the sector to the disk, enter [CTRL-W]. MDDUMP 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
[CR] get existing sector number
'V' verify write

Cursor control
^H backspace
^L move right
^J move down
^K move up

See also MFDUMP - FILE DUMP.

Continued on next page. . .

(7.5 MDDUMP - DISK DUMP continued)

Example:

```
x>MDDUMP
68K PDOS Disk Dump/Alter Utility
  Disk # = 0
  To alter sector, enter "A"; to exit, enter "Z"
  Start Sector = 0
  End Sector = 0

 Sector/Disk=$0000 (0)/0
 000-00F   53 41 47 45 20 50 44 4F 53 20 32 2E 36 64 00 00 FORCE PDOS 3.1..
 010-01F   09 40 00 6D 88 00 08 00 00 80 09 40 A5 5A FF FF .@.m.......@%Z..
 020-02F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 030-03F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 040-04F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 050-05F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 060-06F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 070-07F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 080-08F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 090-09F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0A0-0AF   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0B0-0BF   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0C0-0CF   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0D0-0DF   FF F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .p..............
 0E0-0EF   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
 0F0-0FF   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
  To alter sector, enter "A"; to exit, enter "Z"
  Start Sector = 18
  End Sector = 18

 Sector/Disk=$0012 (18)/0
 000-00F   00 13 00 00 FF FF FF FF 00 00 0D 0E 00 00 04 DC ...............\
 010-01F   00 00 00 54 00 00 00 68 23 14 41 4D 41 5A 49 4E ...T...h#.AMAZIN
 020-02F   47 20 50 52 4F 47 52 41 4D 00 00 00 1C 14 53 45 G PROGRAM.....SE
 030-03F   45 44 3D 00 0B 63 07 1A 63 5C 00 2E 07 08 5C 0D ED=..c..c\....\.
 040-04F   17 4E 06 63 00 00 08 63 06 5C 0D 17 4E 00 1C 14 .N.c...c.\..N...
 050-05F   57 48 41 54 20 41 52 45 20 59 4F 55 52 20 57 49 WHAT ARE YOUR WI
 060-06F   44 54 48 20 41 4E 44 20 4C 45 4E 47 54 48 00 0A DTH AND LENGTH..
 070-07F   64 0A 65 00 23 14 50 4C 45 41 53 45 20 57 41 49 d.e.#.PLEASE WAI
 080-08F   54 2E 2E 2E 2E 00 0B 00 10 64 5C 01 30 65 5C 01 T........d\.0e\.
 090-09F   30 18 66 0A 64 5C 01 30 65 5C 01 30 18 67 0A 64 0.f.d\.0e\.0.g.d
 0A0-0AF   65 32 17 68 00 00 08 69 06 5C 00 07 08 6A 06 5C e2.h...i.\...j.\
 0B0-0BF   00 07 08 6B 06 60 64 32 5C 01 30 17 40 00 08 6C ...k.`d2\.0.@..l
 0C0-0CF   06 5C 01 07 08 6B 5C 01 18 67 06 6C 07 08 6C 06 .\...k\..g.l..l.
 0D0-0DF   6C 5C 01 30 07 08 6D 06 6B 07 08 6E 06 5C 01 00 l\.0..m.k..n.\..
 0E0-0EF   06 6F 06 5C 01 01 64 07 06 70 06 5C 01 01 65 00 .o.\..d..p.\..e.
 0F0-0FF   08 6F 70 18 66 06 5C 01 00 00 1F 70 07 1F 6F 00 .op.f.\....p..o.
  To alter sector, enter "A"; to exit, enter "Z"
  Start Sector =_
```

7.6 MDISKS - DISK NAME LIST

```
        Name: MDISKS
    Function: List available disks
      Format: >MDISKS
              >MDISKS {/}<listtype>
```

Restrictions: PDOS Winchester standard must be implemented.

Description:

The MDISKS utility gives a list of disks that are on-line and their locations. One of three types of lists can be selected by specifying the optional <listtype> parameter. Legal parameters are 'LABEL', 'FILES', or 'BOOT'. The <listtype> parameter can be preceded by a slash (/), in which case only the first character is checked. For example, to select the 'FILE' type of listing you can type 'MDISKS /FILE[CR]', 'MDISKS /F[CR]', or 'MDISKS F[CR]'. If you type MDISKS without a parameter, it will default to the LABEL parameter which provides a short list of disks on-line.

```
x>MDISKS
PDOS 68000 Disk Name Lister
 2 WINI #2           3 SKY WARR/CPU-20   4 Stride 400 3.0b    5 MVME130 PDOS 3.1
 6 MVME133 PDOS 3.1  7 F77 2.2a          8 SY$DSK             9 Force CPU-2 FS
10 FS TERST         12 7120 MPDOS20/81  13 EMS CPU-2RT 3.0b 14 Mizr 9100 3.0b
15 Mizr 7100 3.0b   16 VME/10 3.0b      17 MVME117 3.0b     18 MVME 120 3.0b
x>_
```

The FILES parameter displays disk file and space as well as a bad sector count for each disk on-line.

```
x>MDISKS /FILES
PDOS 68000 Disk Name Lister
Disk Label           # of Files     Sectors     Free        Bad     # of PDOS
                     Curr/Total    Used/Alloc Total/Cont  Sectors   Sectors
 2   WINI #2         1214/1536    36798/39659   752/752      0        40623
 3   SKY WARR/CPU-20   13/64       2148/2148    202/202      0         2360
 . . .
19   FORCE CPU-4 3.0a  75/128      2032/2063    287/250      0         2368
x>_
```

Continued on next page. . .

(7.6 MDISKS - DISK NAME LIST continued)

The BOOT option displays boot information on each disk.

```
x>MDISKS /BOOT
PDOS 68000 Disk Name Lister
Disk Label              Boot       Load       Boot      SYID    BIOS
                        Sector     Address    Size      Chars   Date
 0   >> Disk error 116
 1   >> Disk error 116
 2   WINI #2           bs=40623   la=$000800 sz=150    id=F2 03/07/86                   bs=boot sector
                       "FORCE CPU-2(FS) BIOS: Clock=PI/T. 03/07/86"             la=load address
 3   SKY WARR/CPU-20   bs=2360    la=$000800 sz=157    id=F2 07/10/86                   id=system ID
                       "FORCE CPU-20/21 BIOS 07/10/86"                  sz=boot size
 . . .
19   FORCE CPU-4 3.0a  bs=2368    la=$000800 sz=140    id=F4 12/06/85
                       "FORCE CPU-4 BIOS: Clock=PI/T. 12/06/85"
x>_
```

7.7 MDLOOK - DISK LOOK

Name: MDLOOK
Function: Look for possible file beginnings
Format: >MDLOOK

Restrictions: None.

Description:

The MDLOOK utility scans a disk for possible first sectors of PDOS files. A first sector would be possible if the second word is null. This sector is the back link. If found, a single line is displayed to your console as a hex and ASCII dump. The [ESC] key returns you to the PDOS monitor.

Note: It is worthwhile to have a line of header information on the first line of your text files so that they can be found with this utility.

This information could be used to recover a disk with the MFSAVE utility.

Example:

```
x>MDLOOK
68K PDOS Disk Look Utility
  Disk # = 0
  Start Sector = 0
  End Sector = 100
Sector 1     :0000 0000 0000 0000 ........................................
Sector 16    :0000 0000 0000 0000 ........................................
Sector 17    :0000 0000 0000 0000 ........................................
Sector 18    :0013 0000 FFFF FFFF ..................T...h#.AMAZING PROGRAM.....
Sector 37    :0000 0000 4D41 534D ....MASM &1:SR,#OBJ/8.IF &0.RC.MSYFL OBJ/8,#&1
Sector 38    :0027 0000 4530 3030 .'..E0000000765A08C004042C3C5000186A052E2D0014
Sector 40    :0029 0000 2A09 4230 .)..*.B01:SR.03/26/84.************************
Sector 45    :002E 0000 4530 3030 ....E0000000785A08C003842C3C5000186A052E2D0014
Sector 47    :0030 0000 2A09 4230 .0..*.B02:SR.03/26/84.************************
Sector 52    :0035 0000 4530 3030 .5..E0000000785A08C003842C3C5000186A052E2D0014
Sector 54    :0037 0000 2A09 4230 .7..*.B03:SR.03/26/84.************************
Sector 59    :003C 0000 4530 3030 .<..E00000007E5A08C003A42C3C5000186A052E2D0014
Sector 61    :003E 0000 2A09 4230 .>..*.B04:SR.03/26/84.************************
Sector 66    :0043 0000 4530 3030 .C..E00000007E5A08C003A42C3C5000186A052E2D0014
Sector 68    :0045 0000 2A09 4230 .E..*.B05:SR.03/26/84.************************
Sector 73    :004A 0000 4530 3030 .J..E0000000805A08C003C42C3C5000186A052E2D0014
Sector 75    :004C 0000 2A09 4230 .L..*.B06:SR.03/26/84.************************
Sector 80    :0051 0000 4530 3030 .Q..E0000000865A08C004042C3C5000186A052E2D0014
Sector 82    :0053 0000 2A09 4230 .S..*.B07:SR.03/26/84.************************
Sector 87    :0058 0000 4530 3030 .X..E0000010AA5A08C006442C3C5000186A052E2D0014
Sector 89    :005A 0000 2A09 4230 .Z..*.B08:SR.03/26/84.************************
Sector 97    :0062 0000 4530 3030 .b..E0000000885A08C004242C3C5000186A052E2D0014
Sector 99    :0064 0000 2A09 4230 .d..*.B09:SR.03/26/84.************************
x>_
```

7.8 MDNAME - DISK NAME

Name: MDNAME
Function: Rename PDOS disks
Format: >MDNAME
>MDNAME <disk #>,<new name>

Restrictions: None.

Description:

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

```
Disk #=
Old Name=PDOS DISK OLD
New Name=
```

An alternate way to rename a disk with this utility is to follow the MDNAME call with two parameters in the command line. The first parameter is the disk number and the second is the desired new disk name. MDNAME outputs the old name and renames the disk with no other action. Notice 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 okay). The new name consists of only those characters preceding the comma.

```
MDNAME 0,NEW NAME
```

Examples:

```
x>LS
Disk=PDOS DISK OLD/0           Files=27/128
Lev  Name:ext      Type  Size  Sect  Date created     Last update

 1   MASM          SY C  88/88 006E  21:51 16-Sep-86 21:52 16-Sep-86
x>MDNAME
68K PDOS Disk Name Utility
  Disk # =0
  Old Name=PDOS DISK OLD
  New Name=PDOS DISK NEW
x>LS
Disk=PDOS DISK NEW/0           Files=27/128
Lev  Name:ext      Type  Size  Sect  Date created     Last update

 1   MASM          SY C  88/88 006E  21:51 16-Sep-86 21:52 16-Sep-86
x>MDNAME 0,NEWER NAME
WAS PDOS DISK NEW
x>LS
Disk=NEWER NAME/0              Files=27/128
Lev  Name:ext      Type  Size  Sect  Date created     Last update

 1   MASM          SY C  88/88 006E  21:51 16-Sep-86 21:52 16-Sep-86
x>_
```

7.9 MDSAVE - RECOVER PDOS DISK

Name: MDSAVE
Function: Recover PDOS disk
Format: >MDSAVE

Restrictions: None.

Description:

The MDSAVE utility is used to recover a disk that might have directory problems. The disk is scanned for possible start sectors of PDOS files. This is indicated by a back link (second word) beginning null.

When the start of a file is found, the sectors are read using the forward links until a null forward link is found. As the sectors are read, they are transferred to the destination disk.

The name mask dictates what the new files will be called. The mask must have asterisks (*) which are replaced by a number that starts at 0 for the first file. Each subsequent file gets a new name.

After the process, these files can be examined to determine if they really are PDOS files.

See also MFSAVE - FILE SAVE

Example:

```
x>MDSAVE
68K PDOS Disk File Recovery Utility
        Source Disk = 0
  Start,End Sectors = 0,80
   Destination Disk = 3
     File Name Mask = M***:SR
File=#M__0:SR/3   Sector=18
File=#M__1:SR/3   Sector=37
File=#M__2:SR/3   Sector=38
File=#M__3:SR/3   Sector=40
File=#M__4:SR/3   Sector=45
File=#M__5:SR/3   Sector=47
File=#M__6:SR/3   Sector=52
File=#M__7:SR/3   Sector=54
File=#M__8:SR/3   Sector=59
File=#M__9:SR/3   Sector=61
File=#M_10:SR/3   Sector=66
```

Continued on next page...

(7.9 MDSAVE - RECOVER PDOS DISK continued)

```
File=#M_11:SR/3   Sector=68
File=#M_12:SR/3   Sector=73
File=#M_13:SR/3   Sector=75
File=#M_14:SR/3   Sector=80
x>LS /3
Disk=WDISK #3/3                       Files=20/1024
Lev  Name:ext      Type      Size     Sect  Date created     Last update
 1   M__0:SR                 19/19    0085 11:34 11-May-86 11:34 11-May-86
 1   M__1:SR                  1/1     0098 11:34 11-May-86 11:34 11-May-86
 1   M__2:SR                  2/2     0099 11:34 11-May-86 11:34 11-May-86
 1   M__3:SR                  5/5     009B 11:34 11-May-86 11:34 11-May-86
 1   M__4:SR                  2/2     00A0 11:34 11-May-86 11:34 11-May-86
 1   M__5:SR                  5/5     00A2 11:34 11-May-86 11:34 11-May-86
 1   M__6:SR                  2/2     00A7 11:34 11-May-86 11:34 11-May-86
 1   M__7:SR                  5/5     00A9 11:34 11-May-86 11:34 11-May-86
 1   M__8:SR                  2/2     00AE 11:34 11-May-86 11:34 11-May-86
 1   M__9:SR                  5/5     00B0 11:34 11-May-86 11:34 11-May-86
 1   M_10:SR                  2/2     00B5 11:34 11-May-86 11:34 11-May-86
 1   M_11:SR                  5/5     00B7 11:34 11-May-86 11:34 11-May-86
 1   M_12:SR                  2/2     00BC 11:34 11-May-86 11:34 11-May-86
 1   M_13:SR                  5/5     00BE 11:34 11-May-86 11:34 11-May-86
 1   M_14:SR                  2/2     00C3 11:34 11-May-86 11:34 11-May-86
x>
```

7.10 MFDUMP - FILE DUMP

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

Restrictions: None.

Description:

The MFDUMP utility outputs a hex and ASCII dump of a PDOS file. If no output file is specified, the console terminal is used. The prompt, "INCREMENT," 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 MFDUMP is the file displacement, two hex characters per byte, and the ASCII characters. If an ASCII character is unprintable, then a period is used. The last line will have Fs after the end-of-file is reached.

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

```
MFDUMP DROPEX,LIST/5,32
MFDUMP MFDUMP
```

See also MDDUMP - DISK DUMP.

Example:

```
x>MFDUMP[CR]
68K PDOS File Dump Utility
  File = PSPELL:DIC[CR]
  Output = [CR]
  Increment = [CR]
0000-000F  0000 1505 0000 0000 0000 0001 0000 004B  ...............K
0010-001F  0000 0135 0000 01C8 0000 01C9 0000 01F1  ...5...H...I...q
0020-002F  0000 0211 0000 0216 0000 0221 0000 0222  ...........!..."
0030-003F  0000 0223 8000 02F9 8000 033D 0000 03B5  ...#...y...=...5
0040-004F  0000 03B6 0000 044B 0000 044C 8000 04B4  ...6...K...L...4
0050-005F  8000 0570 0000 05BA 0000 05F3 0000 060D  ...p...:...s....
0060-006F  0000 0624 0000 062A 0000 062B 0000 062C  ...$...*...+...,
0070-007F  0000 06C6 0000 06C7 0000 06C8 8000 06C9  ...F...G...H...I
0080-008F  0000 0785 0000 0786 0000 0787 0000 0788  ................
...
x>
```

7.11 MFFIND - FIND FILE ACROSS DISKS

```
        Name: MFFIND
    Function: Find files on multiple disks
      Format: >MFFIND
              >MFFIND <mask>{,<outfile>}
```

Restrictions: None.

Description:

The MFFIND utility lets you search for files across multiple disk units.

Mask format: <file>{/<disks>}{/F<date>}{/D<date>}

```
      Where:  <file> = TEMP:E**;@
             <disks> = 0-5/24
              <date> = MN/DY/YR
```

See also:

Monitor command LS
MLDIR - DIRECTORY LIST
MLEVEL - LEVEL DIRECTORY LIST
MORDIR - ALPHABETIZE PDOS DIRECTORY

Example:

```
x>MFFIND
68K PDOS Find File Utility
Mask = CHAP@:@
Output = [CR]

**** Disk Name = WINI 2/2                              Files=162/1024
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 21  CHAP01        TX C     211/211   12:10 20-Aug-84  12:29 08-May-85
 21  CHAP02           C     269/269   12:11 20-Aug-84  12:11 20-Aug-84

**** Disk Name = WINI 6/6                              Files=84/1024
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 21  CHAP01        TX C     211/211   12:10 20-Aug-84  12:29 08-May-85
 21  CHAP02           C     269/269   12:11 20-Aug-84  12:11 20-Aug-84

**** Disk Name = TEMP/11                               Files=10/128
LEV  NAME:EXT      TYPE      SIZE      DATE CREATED    LAST UPDATE

 1   CHAP13        TX       248/248   12:20 20-Aug-84  09:02 23-Jul-85
```

7.12 MFSAVE - FILE SAVE

Name: MFSAVE
Function: Restore files from links
Format: >MFSAVE

Restrictions: None.

Description:

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

The first prompt, 'Source <Disk #,Start Sector> = ', asks for the disk where the recoverable file resides, and its starting sector number. Separate the two numbers with a comma. The starting sector number may be entered in decimal if you know it, or you may type it in hexadecimal by preceding the number with a dollar sign. The starting sector number may have been obtained by a previous directory listing (>LS) or MDDMAP when the directory was good, or by MDLOOK if the directory has gone bad. If you don't have the starting sector number, any sector within the file will do, since MFSAVE can scan backwards to the beginning of the file if the sector given starts in the middle of the file.

Get start sector

The second prompt, "Output <Filename> = " asks for where the new file is to be stored. Generally, it is a good idea to store the new file on a different disk unit than the old file to avoid corrupting the data before it is recovered.

Save on different disk

If the 'Start Sector' has a valid backward link, then MFSAVE asks if an attempt should be made to find the beginning of the file. If you type 'Y', then the backward links are followed until a null link is found. If you type 'N', then MFSAVE 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. MFSAVE cannot determine how many bytes are valid in the last sector. As a result, some unexpected characters may be added at the end of the file. You must clean up the text with the editor.

MFSAVE is exited with an [ESC].

See also MDSAVE - RECOVER PDOS DISK.

Continued on next page...

(7.12 MFSAVE - FILE SAVE continued)

Example:

The help file is located on disk 2, starting at sector $00A9.

```
x>LS HLPTX;0/2
Disk=WINI #2/2                          Files=683/1024
Lev  Name:ext     Type      Size      Sect  Date created     Last update
 0   HLPTX        TX        46/46     00A9 14:43 23-Apr-84 11:27 05-Apr-85
```

This is what the first part of the file looks like.

```
x>SF HLPTX/2
HELP
 For further help, enter 'HE ' followed by one of the following:

       MONITOR         PDOS monitor commands
       FILES           List directory & file types
       BP              Baud port
       CT              Create task
       FS              File slots
              .
              .
              .
```

Now, suppose for some reason you couldn't get the file by conventional methods. Using MFSAVE you can recover the file to a scratch disk.

If you had specified a sector address in the middle of the file, MFSAVE would have asked if it should go back and pick up the rest, as well as go on.

```
x>MFSAVE
68K PDOS File Recovery Utility
  Source <Disk #,Start Sector> = 2,$A9
  Output <Filename> = #TEST/8
  Link Forward = 170
  Link Forward = 171
        .
        .
        .
  Link Forward = 22712
  Link Forward = 22713
  Link Forward = 0
```

Now, the 'destroyed' file is recovered and safe in TEST/8.

```
x>LS ;@/8
Disk=SY$DSK/8                           Files=1/32
Lev  Name:ext     Type      Size      Sect  Date created     Last update
 1   TEST            C      46/46     0005 13:49 23-Aug-85 13:49 23-Aug-85
x>SF TEST/8
HELP
 For further help, enter 'HE ' followed by one of the following:

       MONITOR         PDOS monitor commands
       FILES           List directory & file types
       BP              Baud port
       CT              Create task
       FS              File slots
        .
        .
        .
```

You still will need to edit the file and get rid of any garbage at the end of the file, since MFSAVE does not know where the real end of file was.

7.13 MINIT - INITIALIZE PDOS DISK

Name: MINIT
Function: Initialize disks for PDOS file storage
Format: >MINIT

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, MINIT is independent of the disk controller.

MINIT 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 MINIT are:

1) disk number {/Q}, -- 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, -- {"MAX","BOOT",#}
6) and 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 [CTRL-C] aborts the initialization process and returns to the 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, then the 'SUCCESS!!' message is printed to the console.

If a '/Q' follows the disk number, MINIT only writes every 32nd sector, instead of every sector. (Hence, it executes much faster.)

Continued on next page...

(7.13 MINIT - INITIALIZE PDOS DISK continued)

Total Number of PDOS Sectors (on a 5 1/4 floppy disk)

2528 if the disk will be used only for data.
2336 if the disk has a bootstrap on it.

Total Number of PDOS Sectors (on a hard disk)

Depends on the system configuration. Run xxFRMT according to the directions in the Installation and Systems Management guide for your hardware. Select the 'W' option from the first menu, and 'Disp/Alt PDOS partitions' from the second. The numbers in the rightmost column tell the number of sectors with and without a boot. (To exit xxFRMT enter [ESC] to all queries except the 'Update Param RAM' to which you should answer 'N'.)

If you enter 'MAX' for the sector size, MINIT will determine the maximum number of sectors for a data-only disk and allow you to verify the number. If you enter 'BOOT', MINIT will determine the maximum number of sectors for a boot disk.

Examples:

```
x>MINIT
68K PDOS Disk Initialize Utility
  Disk # = 17
  Sides = 2
  Density = D
  Maximum Directory Size = 128
  Total Number of PDOS Sectors (MAX/BOOT/#) = MAX
  Total Number of PDOS Sectors (MAX/BOOT/#) = 2528
  Disk Name = SCRATCH DISK
INIT:   Disk #17
        Double Sided
        Double Density
        128 Files
        2568 Sectors
DESTROY DISK NAMED 'WORK'? Y
Writing sector 0...2496
INITIALIZATION Successful!
x>_
```

7.14 MINST - MEMORY INSTALL

Name: MINST

Function: Make PDOS aware of additional memory

Format: >MINST

>MINST <low address>,<high address>

Restrictions: Should only be used by system manager, since it is possible to corrupt RAM disks and crash other tasks.

Description:

When PDOS boots, it looks for memory in certain addresses. These locations can be modified by re-building PDOS, but sometimes it is convenient to just plug in additional memory without re-assembling and so forth. If the new memory is not contiguous with the old, PDOS will not see it and the memory cannot be used in the normal fashion. MINST provides a way around this problem.

MINST is also a quick way to recover memory lost with the Free Memory monitor command (>FM -nn). "FM -100" tells PDOS to deallocate 100K of memory from the top of your task area. It is a convenient instruction for setting up a RAM disk, but occasionally you may want to recover the memory for tasking again. You can always re-boot, but MINST provides a way to do it with fewer repercussions.

You may invoke MINST with no parameters, in which case it will scan the memory from address $180000 to $800000 for RAM. It may take a while, so if you know the actual address of the memory you want to reclaim, or if you want restrict MINST to a particular area, specify the starting and ending addresses on the command line. MINST does not prompt interactively for missing commands. The addresses may be given in decimal, or in hex, whichever is more convenient. Specify hex by preceding the number with a dollar sign ($).

Before MINST actually does anything, it displays the starting and ending address it has and asks, "Should I start looking?" to allow you to check for mistakes. If you enter "Y" it will begin searching the address space for RAM. It scans the address space given and if it finds memory, it adds it to the memory allocation bit map. It reports the amount of memory actually found in "pages" where a page is 2048 bytes -- the amount of memory represented by one bit in the memory allocation bit map.

Continued on next page...

(7.14 MINST - MEMORY INSTALL continued)

This free memory is then available for use by PDOS and can be allocated through XGUM calls, XCTB calls, and with the Create Task monitor command (>CT). If the memory is adjacent to the end of current task, that task may extend itself into the newly allocated memory with the Get Memory monitor command (>GM). Just make sure that the memory MINST finds really IS free, and not occupied by any active tasks or RAM disks.

One final note: Some memory cards require additional initialization in order to clear any parity errors that they might have at startup. MINST does not perform any hardware-specific work, and cannot reset a memory board if it has flagged an error. In such a case, refer to the specific hardware manual for the board -- it may be necessary to write additional code of your own to properly initialize the memory for use.

Example:

```
x>MINST $100000,$200000
Start address = $100000
  End address = $200000          Verify search range
Should I start looking?Y
  Found RAM block starting at $100000
512 pages added.
x>_
```

7.15 MLDIR - DIRECTORY LIST

Name: MLDIR
Function: List selected directories with wild cards
Format: >MLDIR
>MLDIR {#}<file name mask>,<outfile>

Restrictions: None.

Description:

The MLDIR utility lists selected files from the disk directory. If the program is invoked without parameters, it prompts for the file name mask and the output file name; otherwise it uses the command line parameters.

MLDIR performs essentially the same function as the List Directory monitor command (>LS) with the following differences:

1. If the file name mask is preceded by a pound sign (#), the list of file names is abbreviated to just name and size.
2. MLDIR prints a summary at the end of the listing to tell you the total number of files displayed and their cumulative size.
3. MLDIR allows you to specify a "From" and/or "To" date by appending dates in the form "/Fmm/dd/yy" and/or "/Tmm/dd/yy" to the end of the file specification. This feature lets you look at just those files with modification dates in a certain range.
4. The source code to MLDIR is provided to the user. So, you can customize this program for your needs.

With the exception of the optional pound sign (#) at the beginning and the optional "From" and "To" date fields at the end, the file specification mask is the same as that used elsewhere in PDOS utilities. Either a multiple character wild-card (@) or a single-character wild-card (*) may be used in either the filename or extension fields. Unless specified, the level automatically defaults to all levels. The disk is the current disk, unless specified.

Examples:

Find all files with a ":C" extension on disk 12 with modification dates in the range From 1-Jun-85 To 1-Jul-85. Create the list in short format and output it to LIST/6.

>MLDIR #@:C/12/F6/1/85/T7/1/85,LIST/6

Continued on next page...

(7.15 MLDIR - DIRECTORY LIST continued)

```
>SF LIST/6

  Disk Name = C 1.2B/12                                    Files=63/64
LEV  NAME:EXT       TYPE       SIZE      DATE CREATED     LAST UPDATE

LOCATE:C;12,17
FDIFF:C;152,28
GREP:C;152,52
HANOI:C;152,11
SORTC:C;152,34
WC:C;152,10
  Total for Files Retrieved:  Files=6,   Used/Alloc=152/152
```

Run MLDIR without parameters. Request a short-form list of all files with ":C" extension in level 12 on disk 12. Let the list come to the console.

```
>MLDIR
68K PDOS List Directory Utility
  Mask = #@:C;12/12
  Output =
  Disk Name = C 1.2B/12                                    Files=63/64
LEV  NAME:EXT       TYPE       SIZE      DATE CREATED     LAST UPDATE

CC:C;12,29
CLINK:C;12,22
LOCATE:C;12,17
ROMLINK:C;12,22
TESTXLIB:C;12,91
  Total for Files Retrieved:  Files=5,   Used/Alloc=181/181
```

Get a long form list of all the files in level 12, disk 12 with ":C" extensions. Let the list come to the console.

```
>MLDIR @:C;12/12,,
  Disk Name = C 1.2B/12                                    Files=63/64
LEV  NAME:EXT       TYPE       SIZE      DATE CREATED     LAST UPDATE

 12  CC:C           TX C       29/29    13:17 22-May-85  14:32 08-Jul-85
 12  CLINK:C        TX C       22/22    16:08 20-May-85  14:38 08-Jul-85
 12  LOCATE:C       TX C       17/17    13:05 24-Jun-85  13:26 24-Jun-85
 12  ROMLINK:C      TX C       22/22    13:17 10-Jul-85  13:17 10-Jul-85
 12  TESTXLIB:C     TX C       91/91    09:21 21-Jun-85  13:33 08-Jul-85
  Total for Files Retrieved:  Files=5,   Used/Alloc=181/181
```

See also:

List Directory monitor command (>LS)
MLDIR - DIRECTORY LIST
MLEVEL - LEVEL DIRECTORY LIST
MFFIND - FIND FILE ACROSS DISKS
MORDIR - ALPHABETIZE PDOS DIRECTORY

7.16 MLEVEL - LEVEL DIRECTORY LIST

```
        Name: MLEVEL
    Function: Give a short listing of directories sorted by level
      Format: >MLEVEL
              >MLEVEL <disk #>{,<outfile>}
```

Restrictions: None.

Description:

The MLEVEL 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 MLEVEL in the command line. Otherwise, the utility prompts you for the disk number and the output file name (a carriage-return directs the output to the console).

Examples:

```
x>MLEVEL[CR]
68K PDOS List Directory by Level Utility
  Disk # = 0[CR]
  Output = [CR]
  Disk name = WORK DISK #20/0[CR]
  Files = 78/160[CR]
  Level Files

 0    BOOT,MASM,MEDIT,NEW,SDOS1000,SMAP1000

 1    BENCH:TX,BUG,DO,DO1,DOB,JUNK,PLIST1,PNETR,PNETS,PROJECT,PSPELL
      PSPELL:DIC,SCHEDULE,SNOW,SPRINT2,SURVEY,TEMP,WARD73D

 2    B01:SR,B02:SR,B03:SR,B04:SR,B05:SR,B06:SR,B07:SR,B08:SR,B09:SR
      B10:SR,BDV:SR,MPDOSB:SR,MPDOSK:SR,MPDOSL:SR,MPNETR:SR,MPNETS:SR
      MROB:SR,MROBX:SR,PNETR:SR,PNETS:SR,PSPELL:SR,TASK1:SR,TASK2:SR
      TASK3:SR,TASK4:SR

 10   B00,B01,B02,B03,B04,B05,B06,B07,B08,B09,B10

 18   MBROB,MOVE,MOVEX,MROBX,PDEMO,RDEMO

 50   TASK1,TASK2,TASK3,TASK4

 73   CAL:M84,ELECTRIC:MB,MBASE,MDATA1,MDATA2,MERIT,SCOUTS,SPRINT
x>_
```

See also:

List Levels monitor command (>LL)
List Directory monitor command (>LS)
MLDIR - DIRECTORY LIST
MFFIND - FIND FILE ACROSS DISKS
MORDIR - ALPHABETIZE PDOS DIRECTORY

7.17 MLIB - LIBRARY FILE MANAGER

Name: MLIB
Function: Manage object modules within library files
Format: >MLIB {filename}{,#sect}

Restrictions: Uses large temporary disk file called MLIB:TMP.

Description:

MLIB is a complete library file manager which facilitates the creation of new QLINK library files, just like MLIBGEN. In addition, MLIB handles the merging of library files, along with adding, deleting and replacing of object modules within a library file. MLIB can list the names and XDEF labels of the object modules in a library, either to the screen or to a file. Finally, MLIB allows you to extract modules from a library to a PDOS object file.

MLIB is invoked with two optional parameters, {filename} and {#sect}. {filename} is the name of a new or existing library file that you want to work on. {#sect} is the size of the temporary file that MLIB needs to define, namely MLIB:TMP. The default size is either 100 sectors or the size of the input library file, whichever is larger. The label header portion of the library is kept in memory while the object code portion of the library is written to the MLIB:TMP file. If MLIB:TMP is defined too small for your library, the library setup will fail on PDOS error 56.

If you don't specify a library file name, MLIB prompts you for an input file. To create a library from scratch, just enter [CR], for no input file. After defining the temporary file and processing the input library file, MLIB enters the command menu.

The command line entry is similar to other PDOS utilities, where there is a command character or word, a blank, and the parameters separated by commas. MLIB accepts either the entire command word, the first character, or the first four characters of the command. If you don't specify required parameters, MLIB will prompt you for them one at a time. After manipulating the library file you must write out the changed file using the OUT command, or the session will be lost. If you try to QUIT before writing OUT the last changes, MLIB warns you and requests verification. Enter a carriage return to see the help message describing the MLIB commands and syntax.

Calling sequence:
MLIB {filename}{,#sect} where filename is new or existing library

Continued on next page. . .

(7.17 MLIB - LIBRARY FILE MANAGER continued)

The commands of MLIB are described below:

L,LIST {outfile} Short list of files in library {to file}

The LIST command prints the original file names of the object modules in the library, in the actual order they are stored. Just the names are printed, with no ID information or dates. The names are listed in columns and the list can be paused for viewing by hitting any key. To direct the file list to a printer or file, enter 'TTA' or <filename> as the {outfile} parameter.

X,XDEF {outfile} List of all XDEF labels in library {to file}

The XDEF command prints the original file name for each object module in the library as well as any IDNT information in the file, followed by a condensed list of all of the XDEF external labels. The labels are listed as a decimal section number and colon (if any), label name (up to nine characters), and are separated by commas. The list can be paused for viewing by hitting a key. To direct the file list to a printer or file, just enter 'TTA' or <filename> as the {outfile} parameter.

A,ADD file1 Add file1 to library at end

The ADD command processes an existing object file from disk onto the end of the working library. MLIB reports if there are no XDEF labels in the ADD file and ignores the ADD command. ADD does not check to see if there is another module with the same name already in the library, so be careful. A successful ADD sets the altered flag for QUIT.

D,DELETE file Delete file from library

The DELETE command removes the XDEF labels from the working library and deletes the object code of the module from the temporary file. MLIB alerts you if the module requested is not in the library. A successful DELETE sets the altered flag for QUIT.

R,REPLACE file1,file2 Replace file1 with disk file2

The REPLACE command looks for both the module {file1} in the library and the disk file {file2} on the disk. If MLIB finds both, the the original module is deleted and the new object file is added to the end of the working library file. If one or both are not found, then the REPLACE command is ignored. This command differs from a DELETE and ADD sequence because BOTH file names are checked first before the library is altered. A successful REPLACE sets the altered flag for QUIT.

```
Commands:
L,LIST {outfile}
   Short list of files in library {to file}
X,XDEF {outfile}
   List of all XDEF labels in library {to file}
A,ADD file1
   Add file1 to library at end
D,DELETE file
   Delete file from library
R,REPLACE file1,file2
   Replace file1 with disk file2
C,COPY file1,file2
   Write object from file1 to disk file2
M,MERGE library
   Add modules and labels from library
O,OUT {outfile}
   Build and write out altered library file
Q,QUIT
   Exit without writing
```

Continued on next page. . .

(7.17 MLIB - LIBRARY FILE MANAGER continued)

C,COPY file1,file2 Write object from file1 to disk file2

The COPY command writes out a module from the working library to a disk object file. This extraction process is the complement of the build library process. MLIB tells you if either the library module name or disk file name cannot be found.

M,MERGE library Add modules and labels from library

The MERGE command adds all the object code and labels from another library file into the working library. The object is added to the end of the current library. A successful MERGE sets the altered flag for QUIT.

O,OUT {outfile} Build and write out altered library file

The OUT command builds the label header portion (required by QLINK) of the working library, writes it to the {outfile}, and then copies the object code from the temporary file to {outfile}. This command may take some time for large libraries, since the bulk of the code must be read from MLIB:TMP and then written to {outfile}. A successful OUT resets the altered flag for QUIT.

Q,QUIT Exit without writing

The QUIT command exits the MLIB program and warns you if the working file has been altered but not written out. It closes the working file, MLIB:TMP.

Examples:

```
x>SF L01:SR

*       L01:SR
*
        XDEF    L01,L01A,L01B
        XREF    L04
L01     DC.L    *-L04
L01A    EQU     $10A
L01B    EQU     $10B
        END

x>SF L02  :SR

*       L02:SR
*
        XDEF    L02
        XREF    L01
L02     DC.L    *-L01
        END
```

Continued on next page. . .

(7.17 MLIB - LIBRARY FILE MANAGER continued)

```
x>SF L03:SR

*       L03:SR
*
        XDEF    L03
        XREF    L02
L03     DC.L    *-L02
        END

x>SF L04:SR

*       L04:SR
*
        XDEF    L04,L04AXXXXX01
        XREF    L03
L04     DC.L    *-L03
        SECTION 15
L04AXXXXX23     NOP
        NOP
        NOP
        END


x>MLIB
68K LIBRARY GENERATOR
  Defining workfile MLIB:TMP;199,100
  LIBRARY Filename=[CR]
Lib>ADD L01:OBJ
Lib>ADD L02:OBJ
Lib>ADD L03:OBJ
Lib>ADD L04:OBJ
Lib>LIST
  68K Library Generator  3.01   15:06 05/13/86
    # of Modules=4  # of Labels=7  Workfile size =$00000194
    Modules in Library
L01:OBJ        L02:OBJ        L03:OBJ        L04:OBJ
Lib>XDEF
  68K Library Generator  3.01   15:06 05/13/86
    # of Modules=4  # of Labels=7  Workfile size =$00000194
    XDEF Labels in Library
Filename        XDEF'd labels
L01:OBJ         0:L01,L01A,L01B
L02:OBJ         0:L02
L03:OBJ         0:L03
L04:OBJ         0:L04,15:L04AXXXXX
```

Continued on next page. . .

(7.17 MLIB - LIBRARY FILE MANAGER continued)

```
Lib>DELETE L02:OBJ
Lib>X
  68K Library Generator
    # of Modules=3  # of Labels=6  Workfile size =$00000146
    XDEF Labels in Library
Filename        XDEF'd labels
L01:OBJ         0:L01,L01A,L01B
L03:OBJ         0:L03
L04:OBJ         0:L04,15:L04AXXXXX
Lib>ADD L02:OBJ
Lib>X
  68K Library Generator
    # of Modules=4  # of Labels=7  Workfile size =$00000194
    XDEF Labels in Library
Filename        XDEF'd labels
L01:OBJ         0:L01,L01A,L01B
L03:OBJ         0:L03
L04:OBJ         0:L04,15:L04AXXXXX
L02:OBJ         0:L02
Lib>OUT LIB
Lib>QUIT


x>MLIB LIB
68K LIBRARY GENERATOR
  Defining workfile MLIB:TMP;199,100
  MLIB:TMP already defined. May not be large enough.
  Processing Library file LIB ......
Lib>XDEF
  68K Library Generator
    # of Modules=4  # of Labels=7  Workfile size =$00000194
    XDEF Labels in Library LIB
Filename        XDEF'd labels
L01:OBJ         0:L01,L01A,L01B
L03:OBJ         0:L03
L04:OBJ         0:L04,15:L04AXXXXX
L02:OBJ         0:L02
Lib>DELETE L04:OBJ
Lib>QUIT
  ** WARNING ** Library is altered, but not written.
  Quit anyway (Y/N)? N
Lib>OUT LIB
Lib>QUIT
x>_
```

See also:

MLIBGEN - LIBRARY GENERATOR
QLINK - PDOS QUICK LINKER (Chapter 6)

7.18 MLIBGEN - LIBRARY GENERATOR

Name: MLIBGEN
Function: Combine object files into a single library file
Format: >MLIBGEN

Restrictions: MLIBGEN only builds new libraries. Existing libraries can be edited with MLIB.

Description:

MLIBGEN allows object files to be combined into a single library file. The entry (XDEF) labels for each library object are stored in the header of the library file along with the originating object file name and position of the library object within the library file.

When you specify a library load with the LIBRARY command during QLINK. PDOS will scan your files for any entry symbols that match any unresolved external (XREF) symbols in the link map. If a match occurs, then only the code corresponding to the XDEF label of the single library object is loaded. Thus, only those objects which resolve external symbols will be loaded.

Every time a library object is loaded, the LIBRARY command will start from the beginning of library header and scan for new entries. It continues until no additional matches are found in the link map and library header.

Example:

```
x>MLIBGEN
68K LIBRARY GENERATOR
Copyright 1983-1986, ERII
LIBRARY FILE=YOURLIB:LIB                 The name of your library file
INPUT FILE=SUB1:OBJ                      Origination object files to
INPUT FILE=SUB2:OBJ                        become library objects
INPUT FILE=[CR]                          Type [CR] to end input files
ANY MORE FILES (Y/N)?N                   Enter 'Y' to continue; 'N' to quit.
```

See also:
MLIB - LIBRARY FILE MANAGER
QLINK - PDOS QUICK LINKER (Chapter 6)

7.19 MORDIR - ALPHABETIZE PDOS DIRECTORY

Name: MORDIR
Function: Alphabetize and compress disk directory
Format: >MORDIR
>MORDIR <disk #>{/L}

Restrictions: MORDIR rewrites directory sectors; errors may destroy file information!

Description:

The MORDIR 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, MORDIR prompts you for an alternate sector number to write the directory to. then using the alter mode of MDDMAP, you can reconstruct the directory later (possibly after reformatting the header track).

Normally, MORDIR sorts the directory entries by name only. However, if you wish to sort them only within levels, and sort the levels as well (use the level as a major sort key and the name as a minor sort key) then specify "/L" after the disk number. MORDIR will then arrange the files to be ordered within their levels and the levels themselves to be ordered.

There are two reasons to order the directory. One is to simply organize things so that the file names are more easily read. The second is to speed access to important files. When PDOS needs to find a file, it searches the directory sequentially. If a filename is at the beginning of a directory with 1000 names, it will be found faster than a file at the end of the directory. Good practice is to put frequently accessed files in low-numbered levels and then sort the directory using the '/L' switch.

Example:

```
x>MORDIR
68K PDOS Order Disk Directory Utility
  Disk # = 0
........ Rewrite Directory
x>_
```

Continued on next page. . .

(7.19 MORDIR - ALPHABETIZE PDOS DIRECTORY continued)

```
x>LS ;@/8
Disk=SY$DSK/8                        Files=15/64
Lev  Name:ext     Type     Size      Sect  Date created     Last update
 2   C068         SY C    226/226    0009 14:40 20-May-85 14:40 20-May-85
 2   C168         SY C    200/200    00EB 15:08 20-May-85 15:08 20-May-85
 3   CEND:O       OB C      1/1      01B3 12:46 14-Jun-85 12:46 14-Jun-85
 2   CPP          SY C     74/74     01B4 12:29 20-May-85 14:08 20-May-85
 3   CSTART:O     OB C      5/5      01FE 12:49 26-Jun-85 14:38 16-Jul-85
 2   LOCATE       SY C     15/15     0203 07:43 16-Jun-85 07:43 16-Jun-85
 1   MASMC        SY C     73/73     0212 14:46 15-May-85 14:59 15-May-85
 1   QLINKC       SY C     43/43     025B 16:18 14-May-85 11:06 17-May-85
 3   STDLIB          C    152/152    0286 15:56 10-Jul-85 16:09 10-Jul-85
 2   TRANS68      SY C     46/46     031E 12:04 026-Jun-85 12:50 26-Jun-85
 3   XLIB            C     57/57     034C 09:27 10-Jul-85 09:28 10-Jul-85
 99  CTEMP1:O        C    100/100    0385 22:10 25-Jul-85 22:10 25-Jul-85
 99  CTEMP1:SR2      C    100/100    03E9 22:10 25-Jul-85 22:10 25-Jul-85
 99  CTEMP1:SR1      C     10/10     044D 22:10 25-Jul-85 22:10 25-Jul-85
 99  CTEMP1:L        C     10/10     0457 22:10 25-Jul-85 22:10 25-Jul-85

x>MORDIR 8/L
x>LS ;@/8
Disk=SY$DSK/8                        Files=15/64
Lev  Name:ext     Type     Size      Sect  Date created     Last update
 1   MASMC        SY C     73/73     0212 14:46 15-May-85 14:59 15-May-85
 1   QLINKC       SY C     43/43     025B 16:18 14-May-85 11:06 17-May-85
 2   C068         SY C    226/226    0009 14:40 20-May-85 14:40 20-May-85
 2   C168         SY C    200/200    00EB 15:08 20-May-85 15:08 20-May-85
 2   CPP          SY C     74/74     01B4 12:29 20-May-85 14:08 20-May-85
 2   LOCATE       SY C     15/15     0203 07:43 16-Jun-85 07:43 16-Jun-85
 2   TRANS68      SY C     46/46     031E 12:04 26-Jun-85 12:50 26-Jun-85
 3   CEND:O       OB C      1/1      01B3 12:46 14-Jun-85 12:46 14-Jun-85
 3   CSTART:O     OB C      5/5      01FE 12:49 26-Jun-85 14:38 16-Jul-85
 3   STDLIB          C    152/152    0286 15:56 10-Jul-85 16:09 10-Jul-85
 3   XLIB            C     57/57     034C 09:27 10-Jul-85 09:28 10-Jul-85
 99  CTEMP1:L        C     10/10     0457 22:10 25-Jul-85 22:10 25-Jul-85
 99  CTEMP1:O        C    100/100    0385 22:10 25-Jul-85 22:10 25-Jul-85
 99  CTEMP1:SR1      C     10/10     044D 22:10 25-Jul-85 22:10 25-Jul-85
 99  CTEMP1:SR2      C    100/100    03E9 22:10 25-Jul-85 22:10 25-Jul-85
x>_
```

See also:

List Directory monitor command (>LS)
MLDIR - DIRECTORY LIST
MLEVEL - LEVEL DIRECTORY LIST
MFFIND - FIND FILE ACROSS DISKS

7.20 MPATCH - APPLY A PROGRAM UPGRADE PATCH

```
       Name: MPATCH
   Function: Apply a program patch to OB or SY files not distributed
             in source
     Format: >MPATCH
             >MPATCH <oldfile>,<newfile>,<patchfile>

Restrictions: None.
```

Description:

The MPATCH utility provides a mechanism where PDOS object modules and utilities which are not distributed in source (SRC) form can be easily upgraded in the field. The <patchfile> is typed in by the user from written instructions supplied by the manufacturer. MPATCH will perform CRC-16 checks on both old and new versions to insure that the patch is applied properly.

Example:

(MPATCH instructions are put into the file "PATCH1" using MEDIT).

```
x>RN QLINK,QLINK:OLD
x>MPATCH QLINK:OLD,QLINK,PATCH1
OLDFILE=QLINK:OLD
NEWFILE=QLINK
PATCHFILE=PATCH2
x>_
```

7.21 MSREC - BUILD S-RECORDS

Name: MSREC
Function: Build S-Records from SY or OB file
Format: >MSREC
>MSREC <parameters>

Restrictions: Entire file image must fit in task's memory.
May not be re-entered with >GO command.

Description:

The MSREC utility creates Motorola S-record files from PDOS 'OB' object or 'SY' binary files. You must first enter the file name of the 'OB' or 'SY' input file, and then the file name of the S-record output file. Finally, enter the hexadecimal load and entry address for the S-record file. MSREC then converts the PDOS input file data to S1, S2, or S3 records with checksums. The last record is either an S9, S8, or S7 entry address, which is set equal to the load address that you entered.

<Parameters> are 'NULL' or 'N', 'QUAD' or 'Q', and 'SPLIT' or 'S'. If more than one parameter is used, they should be separated with a slash (/) and may be entered in any order.

The S-record lines are terminated with a [LF][CR], so they are compatible with the normal PDOS file utilities. However, some third party downloading software or firmware REQUIRES that S-records be terminated with [CR][LF][NULL]. To create S-record files that can be downloaded to those systems, MSREC will terminate lines with [CR][LF][NULL] if you pass the parameter 'NULL' when invoking MSREC (MSREC NULL). Don't assume that you need this feature; only try it if the normal S-record output does not download properly.

The 'SPLIT' parameter causes MSREC to output even and odd files with :MXE and :MXO extensions. The 'QUAD' parameter causes MSREC to output four files--upper, upper mid, lower mid and lower byte--with the following extensions respectively: :MX3, :MX2, :MX1, :MX0. It is inconsistent to use the 'QUAD' and 'SPLIT' parameters at the same time.

Examples:

Input file is OB type, output address is 0, lines are terminated with $0A0D, or [LF][CR], normally.

```
x>MSREC
68K PDOS Build S-Record File
  SY or OB Input file=T:OB
  S-record Output file=TEMP
  S-record base address & entry=0
x>SF TEMP
S1210000A08C0010A07CA056A08C002253816EF8A00E0A0D456E7465722068656C6C80
S11A001E6F277320746F207072696E743A00202048656C6C6F2E00D2
S9030000FC
```

Continued on next page. . .

Output null line terminated S-records split into four files:

```
x>MSREC NULL/QUAD
x>MSREC Q/N
```

Output PDOS terminated lines into even and odd split files:

```
x>MSREC SPLIT
x>MSREC S
```

(7.21 MSREC - BUILD S-RECORDS continued)

```
x>MFDUMP TEMP
0000-000F  5331 3231 3030 3030 4130 3843 3030 3130  S1210000A08C0010
0010-001F  4130 3743 4130 3536 4130 3843 3030 3232  A07CA056A08C0022
0020-002F  3533 3831 3645 4638 4130 3045 3041 3044  53816EF8A00E0A0D
0030-003F  3435 3645 3734 3635 3732 3230 3638 3635  456E746572206865
0040-004F  3643 3643 3830 0A0D 5331 3141 3030 3145  6C6C80..S11A001E
0050-005F  3646 3237 3733 3230 3734 3646 3230 3730  6F277320746F2070
0060-006F  3732 3639 3645 3734 3341 3030 3230 3230  72696E743A002020
0070-007F  3438 3635 3643 3643 3646 3245 3030 4432  48656C6C6F2E00D2
0080-008F  0A0D 5339 3033 3030 3030 4643 0A0D FFFF  ..S9030000FC....
```

Input file is SY type, base address is $1000, and lines are terminated with $0D0A80, which PDOS outputs as [CR], [LF], [NUL].

```
x>MSREC NULL
68K PDOS Build S-Record File (w/nulls)
  SY or OB Input file=T
  S-record Output file=TEMP
  S-record base address & entry=1000
x>SF TEMP
S1211000A08C0010A07CA056A08C002253816EF8A00E0A0D456E7465722068656C6C70
S11A101E6F277320746F207072696E743A00202048656C6C6F2E00C2
 S9031000EC
x>MFDUMP TEMP
0000-000F  5331 3231 3130 3030 4130 3843 3030 3130  S1211000A08C0010
0010-001F  4130 3743 4130 3536 4130 3843 3030 3232  A07CA056A08C0022
0020-002F  3533 3831 3645 4638 4130 3045 3041 3044  53816EF8A00E0A0D
0030-003F  3435 3645 3734 3635 3732 3230 3638 3635  456E746572206865
0040-004F  3643 3643 3730 0D0A 8053 3131 4131 3031  6C6C70...S11A101
0050-005F  4536 4632 3737 3332 3037 3436 4632 3037  E6F277320746F207
0060-006F  3037 3236 3936 4537 3433 4130 3032 3032  072696E743A00202
0070-007F  3034 3836 3536 4336 4336 4632 4530 3043  048656C6C6F2E00C
0080-008F  320D 0A80 5339 3033 3130 3030 4543 0D0A  2...S9031000EC..
0090-009F  80FF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ...............
```

Input file is SY type, base address is $300800, which causes an S2 type data record and an S8 type entry record.

```
x>MSREC
68K PDOS Build S-Record File
  SY or OB Input file=T
  S-record Output file=TEMP
  S-record base address & entry=300800
x>SF TEMP
S222300800A08C0010A07CA056A08C002253816EF8A00E0A0D456E7465722068656C6C47
S21E30081E6F277320746F207072696E743A00202048656C6C6F2E00E9000AA3
S804300800C3
x>_
```

7.22 MSYFL - BUILD SY OBJECT FILE

```
        Name: MSYFL
    Function: Build PDOS SY object file
      Format: >MSYFL
              >MSYFL <src file>,<SY file>

Restrictions: Reads entire file into memory.
```

Description:

The MSYFL utility builds an 'SY' type object file from either a PDOS 'OB' file or a Motorola S-record text file. 'SY' files are position independent, memory image files.

Example:

```
x>ASM PSPELL
x>MASM PSPELL:SR,#OBJ/8
68000 PDOS Assembler
ERII, Copyright 1983-1986
SRC=PSPELL:SR
OBJ=#OBJ/8
LST=
ERR=
XRF=

END OF PASS 1
END OF PASS 2
x>IF .RC
x>MSYFL OBJ/8,#PSPELL
68K PDOS SY File Maker Utility
  Source file = OBJ/8
  Destination File = #PSPELL
   SECTION LENGTH = E000000CE2
    Entry Address = 00000000
x>RC
x>_
```

7.23 MSYOB - SYFILE TO OBJECT

Name: MSYOB
Function: Convert SYfile to PDOS tagged object format
Format: >MSYOB
>MSYOB <src file>,<obj file>

Restrictions: Does not restore any relocatability to the file.

Description:

MSYOB takes a 68000 program in memory image format (SYfile) and converts it to a PDOS tagged object format file. This might be necessary to merge an existing program in with other code via the linker. Another possible use is when transferring a binary file over serial lines between two PDOS computers. If the two lines are set to 7 bits of data, the binary information may not transfer properly. A way around the problem is to convert the binary data to an "object" file with MSYOB, send the resulting ASCII file over the serial line, and then convert the object file back to binary with MSYFL.

Example:

The program DC is a small calculator type program in SYfile format.

```
x>LS DC;@
Disk=WINI 2/2                          Files=164/1024
Lev  Name:ext     Type     Size     Sect  Date created    Last update
 98  DC           SY        1/1     0F17 16:13 28-May-85 17:57 26-Jul-85

x>MFDUMP DC
0000-000F  A05A 6206 A08C 0066 A00E A056 2E01 A05A   Zb. ..f . V.. Z
0010-001F  621E 4A2E 042E 6612 2207 A054 0078 A08A  b.J...f.". T.x .
0020-002F  A08C 007D A052 A08A A00E 2007 A03A A00E   ..} R . . . . : .
0030-003F  7C00 0C11 002B 6718 5846 0C11 002D 6710  |....+g.XF...-g.
0040-004F  5846 0C11 002A 6708 5846 0C11 002F 66B4  XF...*g.XF.../f4
0050-005F  A05A 63B0 A056 4EFB 6002 DE81 600C 9E81   ZcO VN{`.^.`...
0060-006F  6008 CFC1 6004 8FC1 48C7 60A2 0A0D 4443  `.OA`..AHG`"..DC
0070-007F  3A20 3C4E 313E 2C3C 2B2D 2A2F 3E2C 3C4E  : <N1>,<+-*/>,<N
0080-008F  323E 2C3C 2B2D 2A2F 3E2C 2E2E 2E2C 3C4E  2>,<+-*/>,...,<N
0090-009F  6E3E 2E00 0D0A 414E 5357 4552 3A20 0020  n>....ANSWER: .
00A0-00AF  4F52 2024 00FF FFFF FFFF FFFF FFFF FFFF  OR $...........
```

Continued on next page...

(7.23 MSYOB - SYFILE TO OBJECT continued)

```
x>MSYOB
68K PDOS OB File Maker Utility
  Source file = DC
  Destination File = DC:OB

x>LS DC:OB
Disk=WINI 2/2                                Files=164/1024
Lev  Name:ext       Type      Size      Sect  Date created     Last update
 1   DC:OB          OB         2/2      0DC7 17:51 26-Jul-85 18:02 26-Jul-85

x>SF DC:OB
E0000000A520000000005A05A62065A08C00665A00EA05652E01A05A5621E4A2E5042E6612F2D
52207A05450078A08A5A08C007D5A052A08A5A00E20075A03AA00E57C000C115002B6718F58
558460C115002D6710558460C115002A6708558460C115002F66B45A05A63B05A0564EFBF69
56002DE815600C9E8156008CFC1560048FC1548C760A250A0D444353A203C4E5313E2C3CF12
52B2D2A2F53E2C3C4E5323E2C3C52B2D2A2F53E2C2E2E52E2C3C4E56E3E2E0050D0A414EF81
55357455253A20002054F522024300FA3
1000000000
:MSYOB  07/26/85 18:02:51
```

Note that OB files are directly executable under PDOS, since the loader knows how to make a memory image from an OB file.

```
x>DC
DC: <N1>,<+-*/>,<N2>,<+-*/>,...,<Nn>.
x>DC:OB
DC: <N1>,<+-*/>,<N2>,<+-*/>,...,<Nn>.
```

7.24 MTERM - SET TASK TERMINAL TYPE

```
      Name: MTERM
  Function: Set terminal cursor functions for task only
    Format: >MTERM
            >MTERM <type>
            >MTERM U,<clear screen>,<cursor pos>,<bias>,<sequence>
            >MTERM U,<clear screen>,,
```

Restrictions: None.

Description:

The MTERM utility sets the position cursor (PSC$) and clear screen (CSC$) variables in the task control block (TCB). This utility makes it easy to use 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 parent task control set. MTERM provides an easy way for a task to change its function characters while the system is running.

If a legal <type> is passed in the command line, then MTERM simply enters the corresponding sequences into the user status block. If type 'U' is selected on the command line, you may define the terminal control sequences in the same order and format as they are entered from the menu. Otherwise, the utility prints the following table of options:

```
68K PDOS Change Terminal Type Utility
Terminals:
    A=ADDS Regent 25
    D=Decscope (VT52)
    H=Hazeltine 1520
    I=Intertube II
    L=Lear Seigler ADM3a
    S=Soroc IQ120
    M=Data Media Excel 12
    V=VT100 / ANSI terminal
    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.

Continued on next page. . .

(7.24 MTERM - SET TASK TERMINAL TYPE continued)

Examples:

```
x>MTERM                                                        Invoke interactively
68K PDOS Change Terminal Type Utility
Terminals:
    A=ADDS Regent 25
    D=Decscope (VT52)
    H=Hazeltine 1520
    I=Intertube II
    L=Lear Seigler ADM3a
    S=Soroc IQ120
    M=Data Media Excel 12
    V=VT100 / ANSI terminal
    U=User Defined
Type = H                                                       Try Hazeltine
x>MTERM
68K PDOS Change Terminal Type Utility
Terminals:
    A=ADDS Regent 25
    D=Decscope (VT52)
    H=Hazeltine 1520
    I=Intertube II
    L=Lear Seigler ADM3a
    S=Soroc IQ120
    M=Data Media Excel 12
    V=VT100 / ANSI terminal
    U=User Defined
Type = U                                                       Let's look at them now...

Enter characters as follows:
  Printable characters:                     #Y = 2359
  Control chars preceded with a "^":      ^A^S = 0113
  ASCII names in brackets:           <ESC><BEL> = 1B07
  ASCII Literals in HEX in brackets:     <1B>Y = 1B59
  BIOS Handler for clear or position:    <0><0>

Clear Screen Characters............<ESC><5C>   New =
Position Cursor Prefix Characters...<ESC><11>   New =
                                                               Reset to Soroc
```

7.25 MTRANS - FILE TRANSFER

Name: MTRANS
Function: Transfer selected files with wild cards
Format: >MTRANS @:@;@/<source disk #>,<dest. disk #>{select string}

Restrictions: None.

Description:

The MTRANS 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. MTRANS is useful in 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 MTRANS to ask whether you want each eligible file transferred or not. Answer 'Y' for yes and 'N' for no. These options can be entered in any order.

Continued on next page. . .

(7.25 MTRANS - FILE TRANSFER continued)

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

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

The default options are equivalent to '/D/U' (i.e., transfer all files, regardless of whether they are defined on the target disk), 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 MTRANS 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 you are 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, MTRANS asks if the transfer should continue with the other files. For instance, there might be an error in transferring a large file because there is not enough room to receive it, but you might want MTRANS to continue transferring the smaller files that will fit.

Continued on next page. . .

(7.25 MTRANS - FILE TRANSFER continued)

A file with the driver attribute "DR" is transferred by modifying the attribute, transferring the file, and restoring the attribute for both the original and the copy. If this were not done, an attempt to read or write the driver file would only access the driver's device instead of copying the driver file itself. A [CTRL-C] stops the transferring of programs after the current transfer is completed.

[CTRL-C] stops transfer

See also:

MBACK — sector by sector copy instead of file by file

Transfer File monitor command (>TF) — memory resident, truncates file to EOF

Examples:

```
x>MTRANS
68K PDOS File Transfer Utility
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]
x>MTRANS 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
x>_
```

7.26 MUNDL - UN-DELETE FILE

Name: MUNDL
Function: Reverses previous delete file process
Format: >MUNDL

Restrictions: Recovery must be made immediately after deletion.
No other users should be accessing the disk.
Description:

When a PDOS file is deleted using XLDF, >DL, or >DM, the first two bytes of the file name in the directory entry is zeroed out. All the sectors allocated to that file are re-allocated (freed up). The MUNDL utility reverses this process to immediately recover mistakenly deleted files right after they are deleted. If you extend or define other files, the newly-freed sectors may be taken from the deleted file's sectors, making it impossible for MUNDL to recover them.

MUNDL prompts for the PDOS disk number where the deleted files were (and still are) located. It then reads in the header sector, the bitmap sectors, and all the directory sectors into memory and operates on them there. Only after you have made all the changes you want and entered a verifying 'Y' reply, does MUNDL write to the disk.

No other users should access the disk during the recover process during the recover process. MUNDL looks at each directory entry for un-deletable candidates. These entries have a null word at the beginning of the file name, a non-null first sector, and the first sector is 'FREE' in the bitmap. If any un-delete entries are found, MUNDL shows you the file name, with 'zz' substituted for the zeroed file name bytes, and asks if you want to un-delete the file. Enter one of two characters of the file name to recover it. Enter a carriage return to skip it.

MUNDL inserts the character(s) into the directory entry, then allocates the sectors in the bitmap, setting their bits to one, meaning 'ALLOCATED.' When the directory has been exhausted, then if any files were un-deleted, MUNDL asks if you want to write the changes to the disk. MUNDL then writes out the header, bitmap, and directory sectors back to the disk.

Continued on next page. . .

(7.26 MUNDL - UN-DELETE FILE continued)

Example:

```
5>LS
Disk=FORCE CPU-1 3.2x/5                Files=78/96
Lev  Name:ext      Type      Size      Sect    Date created     Last update
 1   MASM          SY C      87/87     0026   13:14 16-Jan-85 16:28 17-Oct-86
 1   MEDIT         SY C      26/26     007D   11:38 17-Oct-85 12:44 22-Oct-86
 1   QLINK         SY C
5>MUNDL
68K PDOS FILE UN-DELETER
  Enter Disk # 5
    Disk Name : FORCE CPU-1 3.2x
  Reading in sector bit-map & directory.............Done
    Found File zzART1;1
  Enter 1st 2 chars OR [cr] to skip :ST
  Un-deleting: resetting sectors as allocated..Done
    With NO ruined files, you added 1 files back to the disk.
  Would you like to write updated directory to disk ?Y
  Writing out sector bit-map & directory.............Done
5>LS
Disk=FORCE CPU-1 3.2x/5                Files=79/96
Lev  Name:ext      Type      Size      Sect    Date created     Last update
 1   MASM          SY C      87/87     0026   13:14 16-Jan-85 16:28 17-Oct-86
 1   MEDIT         SY C      26/26     007D   11:38 17-Oct-85 12:44 22-Oct-86
 1   QLINK         SY C      53/53     0097   15:48 25-Apr-84 16:36 17-Oct-86
 1   START1           +C      1/1      00F5   15:13 02-Dec-86 15:14 02-Dec-86
Files=4            Used=167/167
```

7.27 MABORT - TASK ABORTER

Name: MABORT
Function: Aborts task back to the PDOS monitor
Format: >CT MABORT,2
>CT (MABORT <count>),2

Restrictions: This is a background task and NOT a user utility.
Only one MABORT task should be running in a system.
Aborted tasks may leave files opened.

Description:

The MABORT program is a background task that implements an "ABORT TASK" function in PDOS. This function is started by creating the MABORT program as a task using the Create Task (>CT) monitor command. Only 2k bytes are required for the task. MABORT converts the optional parameter (<count>) into the trigger count that is used in breaking tasks. The default value is 2.

Once the MABORT task is running, any user that gets "locked up" can enter successive break characters ([CTRL-C]) on the terminal. The user will be returned to the PDOS monitor by MABORT. Some situations which commonly occur in development from which MABORT can break out include the following:

1. trying to send output to a printer that is non-existent or off-line,
2. endless loops in programs,
3. errors in transparent mode, or
4. bus errors while debugging EXT$(A6) menus.

While monitoring the incoming characters, PDOS looks for certain characters, one of which is called the break character, usually a [CTRL-C]. PDOS keeps track of how many successive break characters are received on each port in a table in SYRAM, CCNT.(A5). The MABORT task, when created, places its task number in the E124.(A5) byte of SYRAM, so that others may know which task is the MABORT task. It raises its own execution priority to 255 and then enters a periodic loop which executes only once a second.

Example:
First, create a task with the default
2-character trigger.

```
x>CT MABORT,2              (create MABORT)
Task #1
x>LOOP                     (run endless loop program)
                          (hit 2 [CTRL-C] characters)
PDOS ERR 85 Aborted task
x>_

x>CT (MABORT 4),2          (create MABORT with 4
                            character trigger)
Task #2
x>_
```

Continued on next page . . .

(7.27 MABORT - TASK ABORTER continued)

When awakened, MABORT sets the task lock (XLKT) and checks the break character counters for all the ports. If any of the ports have a count higher than the trigger number, MABORT looks for the task with that port assigned for input. If found, MABORT sets a flag in the task list that tells the PDOS kernel to exit to the monitor with error 85 (XERR). It also resets some TCB parameters that may have locked up the task:

```
PRT$.B = port #   Reset input port to task's
                        originally allocated
U1P$.B = port #   Reset unit 1 output port to
                        original
UNT$.B = 1        Set to unit 1
ACI$.L = 0        Reset AC file IDs
IMP$.L = 0        Reset input memory pointer
SFI$.W = 0        Set for no spool unit file
ECF$.W = 0        Reset echo flag
SPU$.B = 0        Reset spool unit mask
EXT$.L = 0        Reset XEXT trap pointer
ERR$.L = 0        Reset XERR trap pointer
```

MABORT then goes back to sleep for another second. The monitor reports error 85 to the aborted task. On most PDOS 3.2 standard systems, an MABORT task is created by the SY$STRT autostart file on the BOOT disk.

Some PDOS systems have a 'SOFTWARE ABORT' switch which usually causes a level 7 interrupt. These systems can implement an interrupt service routine in the BIOS which works in conjunction with MABORT to break all tasks. The interrupt service routine loads the break character counters of all the ports to a big number (larger that MABORT trigger count), and then sets the priority of each task (except for the MABORT task) equal to 64.

Continued on next page...

(7.27 MABORT - TASK ABORTER continued)

Example:

The following code is an example of a software abort switch interrupt service routine that could be included in the xxBIOS:SR file to implement a system break function. A corresponding entry should be included in the BINTB table to point the abort switch vector to this routine.

```
**************************************************
*      SOFTWARE ABORT SWITCH:
*        Set all port's break counters to 10 so that
*        if a MABORT task is installed, they will all
*        be aborted to the monitor.  Also set all tasks'
*        priorities to 64.
*
RL       REG      D0/A0/A1
*
ABSW     MOVEM.L  RL,-(A7)         ;SAVE SOME REGS
         MOVEA.L  B$SRAM,A1        ;POINT AT SYRAM
         LEA.L    CCNT.(A1),A0     ;POINT AT ^C COUNTER TABLE
         MOVEQ.L  #16-1,D0
*
@010     MOVE.B   #10,(A0)+        ;SET ALL TO BREAK
           DBF    D0,@010
         LEA.L    TQUE.(A1),A0     ;POINT TO TASK QUEUE
*
@015     TST.W    (A0)             ;MORE?
           BEQ.S  @020             ;N, QUIT
         MOVE.W   (A0)+,D0         ;Y, GET PRIORITY | TASK #
         CMP.B    E124.(A1),D0     ;IS THIS WATCH TASK?
           BEQ.S  @015             ;Y, SKIP PRIORITY SETTING
         MOVE.B   #$40,-2(A0)      ;N, SET TO PRIORITY 64
         BRA.S    @015             ;LOOP
*
@020     MOVEM.L  (A7)+,RL         ;RESTORE
         RTE                       ;AND RTE
```

7.28 WIND1 - CREATE VIRTUAL PORTING TASK

Name: WIND1
Function: Initialize virtual porting
Format: >CT (WIND1,,,),<size>
>CT (WIND1,<window list>,<port list>,<print>,<append>),<size>

Restrictions: If print or append file is used, it must be predefined. Only a file can be used for append output. <size> is equal to number of windows times two plus four. No I/O port should be assigned. XPCR and XPDC bypass window processor - no character update. Special terminal functions not supported by virtual ports. Position cursor and clear screen may require screen refresh.

Description:

PDOS virtual ports (sometimes referred to as "windows") allow selective switching of physical I/O ports to logical task ports. This means that a single terminal can dynamically switch between I/O ports that may be assigned to different tasks or updated by a single task with multiple screen output. A screen image is maintained for all active ports and the switching process updates the terminal with the current display for the selected port.

With PDOS virtual ports, the system acts as if there were more terminals on the system. As a result, multiple tasks are accessible from one terminal.

A high priority virtual port task maintains the screen buffers and handles screen refreshing and buffer printing. A special key sequence is used to switch from one virtual port to another. When a selection is made, PDOS maps your keyboard to another port and the virtual port task clears and updates your display to reflect the current screen.

CT (WIND1,<window list>,<port list>,<print>,<append>),<size>

The virtual port process is set up by creating a task with the WIND1 program. The size of the task is equal to the number of virtual ports times two plus four. No I/O port should be assigned.

If WIND1 encounters an error during its initialization, it will notify its parent task with the appropriate message through the message buffers. Possible errors include:

1. Not enough memory allocated.
2. Virtual port process already executing.
3. Illegal parameters specified.

Continued on next page. . .

(7.28 WIND1 - CREATE VIRTUAL PORTING TASK continued)

WIND1 signals PDOS that virtual porting is now active by setting the SYRAM variables WIND. and WADR., and allocates buffers for the virtual screens. WIND1 sets its execution priority to 100 and kill-protects itself by setting its parent task to -1. The task suspends on event 127.

Virtual ports are selected with a leading control character followed by the port number. (Ports 10 through 15 are selected by letters A through F.) The default control character is [CTRL-X] which is also the clear buffer code. This is alterable at sysgen time by setting B.WND for MBIOS:SR.

[CTRL-X]P sets the print bit (#13) which enables printing. Two consecutive [CTRL-X]s translate to a single [CTRL-X] which is passed through to the input character processor.

The port number external to PDOS is referred to as the physical port. The port number after virtual port translation is referred to as the logical port.

The <window list> parameter specifies the PDOS I/O ports that are to accept virtual access. The ports are specified by number and are separated by slashes (/). Consecutive ports can be specified by separating the first and last port number with a hyphen (-). Default is 1-15 or all PDOS ports.

The <port list> parameter selects those PDOS I/O ports that are permitted to access other ports. This allows some system security for selected ports. The format is the same as the <window list> and the default is for port 1 only.

The third parameter <print> specifies where a screen dump is sent. It may be to a file or an I/O port. Whenever the screen dump function is activated with ([CTRL-X]P), then the WIND1 program opens the <print> file, outputs the current screen image, and closes the file. A dump header with the current time and date precedes the output. If a file is used, it must be pre-defined or defined using the '#' symbol.

The forth parameter <append> is similar to the <print> parameter with the following exceptions:

1) Only a file can be used for output
2) The output is appended to the file which must be pre-defined or auto-defined (#).

See also:

Appendix H - VIRTUAL PORT INTERNALS
Chapter 3 - HOW DO I SET UP VIRTUAL PORTS?

The four parameters for WIND1 are as follows:

```
<window list> = LOGICAL VIRTUAL PORTS (Default=1-15)
  <port list> = PHYSICAL PORTS ALLOWED TO WINDOW
                (Default=1)
      <print> = OUTPUT FILE OR PORT # (Default=none)
     <append> = APPEND OUTPUT FILE (Default=none)

   1/2/3/8/13/14/15 = 7 ports 1,2,3,8,13,14,15
        1-3/8/13-15 = Same as above
```

The following example creates window processing for ports 1, 3, 4, and 5. Only port 1 is allowed to window and a [CTRL-X]P sends a screen image to port #2.

x>CT (WIND1 1/3-5,,2),12

This example creates windows for all 15 PDOS ports. Physical ports 1-4 can window. A [CNTRL-X]P sends a screen image to file PBUF and appends the same image onto file ABUF.

x>CT (WIND1 1-15,1-4,PBUF,ABUF),34

7.29 WKILL - DISABLE VIRTUAL PORTS

Name: WKILL
Function: disables virtual port task
Format: >WKILL {<task #>}

Restrictions: WKILL can only be executed from task 0.

Description:

The window processor (WIND1) contains the screen image buffers and, as such, simply killing the task frees memory to PDOS that would still be written to by the character interrupt processor. The WKILL utility is included to disable virtual port processing.

WKILL {<task #>}

The optional parameter <task #> selects the window processor task if it is not task one. WKILL clears the SYRAM variables WIND. and WADR., unprotects the virtual port processor and executes a KT <task #> to kill the task.

See also:
WIND1 - CREATE VIRTUAL PORTING TASK

7.30 WLOOK - VIEW VIRTUAL PORT PARAMETERS

Name: WLOOK
Function: Displays virtual port parameters
Format: >WLOOK

Restrictions: None.

Description:

The virtual port monitor utility WLOOK displays the screen buffer addresses, the current refresh clear screen/position cursor codes, and then dynamically displays the current port translation table (WIND.). An [ESC] returns to the PDOS monitor.

Example:

```
x>CT (WIND1 1/3-6,1/4),16
*TASK #1
x>WLOOK
WINDOW BUFFERS:

 #1=$000EA23C  #2=Undefined  #3=$000EA9BC  #4=$000EB13C  #5=$000EB8BC
 #6=$000EC03C  #7=Undefined  #8=Undefined  #9=Undefined  #A=Undefined
 #B=Undefined  #C=Undefined  #D=Undefined  #E=Undefined  #F=Undefined

PORT CLEAR/POSITION CODES:

 #1=$AA009B3D  #2=$AA009B3D  #3=$AA009B3D  #4=$AA009B3D  #5=$AA009B3D
 #6=$AA009B3D  #7=$AA009B3D  #8=$AA009B3D  #9=$AA009B3D  #A=$AA009B3D
 #B=$AA009B3D  #C=$AA009B3D  #D=$AA009B3D  #E=$AA009B3D  #F=$AA009B3D

Enter [ESC] to exit to PDOS

 0006 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 8006 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 0001 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 8001 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
 0006 0080 0080 0004 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080 0080
```

See also:
WIND1 - CREATE VIRTUAL PORTING TASK

7.31 WTERM - SET TERMINAL TYPE FOR VIRTUAL PORTS

Name: WTERM
Function: Displays windowing parameters
Format: >WTERM <port #>,<type char>

Restrictions: None.

Description:

The virtual port processor initializes its port position cursor and clear screen codes to those of the parent task. Hence, refresh uses the same codes for all ports unless it is altered by the WTERM utiltity after the window process is executing. These codes are located immediately following the address table (WADR.).

The WTERM utiltity has identical parameter definitions as the PDOS MTERM utility with the exception that the first parameter is a windowing port number. (See 7.24 MTERM - SET TASK TERMINAL TYPE)

Example:

```
x>WTERM 5,S

x>WTERM
68K PDOS Change Terminal Type Utility
Terminals:
    A=ADDS Regent 25
    D=Decscope (VT52)
    H=Hazeltine 1520
    I=Intertube II
    L=Lear Seigler ADM3a
    S=Soroc IQ120
    M=Data Media Excel 12
    V=VT100 / ANSI terminal
    U=User Defined
Port #=6
Type = V
x>_
```

See also:
WIND1 - CREATE VIRTUAL PORTING TASK

7.32 MMKBT - MAKE DISK BOOT

Name: MMKBT
Function: Installs PDOS system and bootstrap onto disks
Format: >MMKBT

Restrictions: May not be re-entered with >GO monitor command.

Description:

The MMKBT utility is used to install bootable PDOS files onto disks. The disk should first be prepared for use with the PDOS system.

MMKBT offers three options. The first option is (F)ile for creating a new boot disk from a file. The second is (M)emory for creating a new boot disk from a memory area. The last option is (B)ootstrap for putting out special sector information onto physical track 0 of the disk. Some systems may not use the bootstrap option. To select any option, type the letter followed by a [CR].

This utility provides defaults which are the correct parameters in most cases. The defaults shown below may not correspond to your hardware system. Consult your Installation and Systems Management guide for specific information about your system.

(F)ILE OPTION

When the (F)ile option is selected, you can install a boot from a file on your disk. You are prompted for the filename. The filename will be read in and sized. You are then asked for the boot size, the load address, the disk number, and the boot sector number. You must confirm the installation before any data is written to the disk.

Continued on next page. . .

```
>MMKBT
68K PDOS Make Boot Disk Utility
(F)ile, (M)emory, or (B)ootstrap? F
        Filename = xxDOS
       Boot size = $00000800
    Load address = $00000800
            Disk = 0
          Sector = 2336

Write 120 sector boot beginning at address
$00000800 with load address $00000800 to disk
0, sector 2336.  Ready (Y/N)?Y
  One moment, Please...
  Wrote out header information.
  PDOS written successfully!!
```

(7.32 MMKBT - MAKE DISK BOOT continued)

(M)EMORY OPTION

The (M)emory option allows you to copy memory onto the boot area of a disk. This is typically done to build a boot image of the currently executing system. After executing the MMKBT program, select the memory option by typing 'M[CR].'

You are asked for the memory start address and end address for the boot. The defaults are determined by your standard system setup. You may select the defaults by typing a [CR] or you may enter new addresses followed by a [CR]. (Remember to use the '$' to prefix hexadecimal numbers). The rest of the questions are the same as the file option.

(B)OOTSTRAP OPTION

The (B)ootstrap option writes out an IPL file to a physical sector of the floppy or Winchester disk. The PDOS IPL file comes from the SY file, xxBOOT, which is generated by the xxBOOT:GEN procedure file.

THE BOOTSTRAP OPTION IS NOT APPLICABLE ON SOME HARDWARE SYSTEMS. Consult your Installation and Systems Management guide for specific information about the bootstrap option.

```
>MMKBT
68K PDOS Make Boot Disk Utility
(F)ile, (M)emory, or (B)ootstrap? M
    Start address = $00001000
      End address = $00000800
     Load address = $00000800
             Disk = 0
           Sector = 2336

Write 112 sector boot beginning at address
$00001000 with load address $00000800 to disk
0, sector 2336.  Ready (Y/N)?Y
  One moment, Please...
  Wrote out header information.
  PDOS written successfully!!
```

7.33 MTIME - SET PDOS/BATTERY CLOCK

Name: MTIME
Function: Set the PDOS and/or battery clocks
Format: >MTIME
>MTIME {P / B /}{,<yr>}

Restrictions: The battery clock may only be set if there is a battery clock chip present in the system.

Description:

MTIME sets the PDOS clock from the battery clock and the battery clock from the PDOS clock. It also allows you to set the year if the battery clock chip doesn't maintain the year. Consult the Installation and Systems Management guide for your specific hardware system.

Either the letter 'P' or the letter 'B' may be omitted from the first parameter. If only the letter 'P' is specified, then the PDOS clock is set from the battery clock.

When the first parameter is the letter 'B,' the battery clock is set to the current PDOS clock values. This is usually done after the PDOS clock has been set with the "ID" monitor command.

```
x>ID
PDOS/68020 R3.2
ERII, Copyright 1983-1986
xxxBIOS
DATE=00-???-00 16-Dec-86
TIME=00:00:01 12:52
x>MTIME B,86
```

When no parameters are specified, the values of both the battery and PDOS clocks are specified.

```
x>MTIME
PDOS CLOCK = 12:55:24 16-Dec-86
BATT CLOCK = 12:55:26 16-Dec-86 Tuesday
```

Some errors may occur if you attempt to run MTIME without a battery clock chip present. Consult your Installation and Systems Management guide.

7.34 xxFRMT - DISK HARDWARE FORMAT

Name: xxFRMT (where xx is the system ID)
Function: Hardware format disks and set up PDOS partitions
Format: >xxFRMT

Restrictions: xxFRMT may be run only when no other tasks are running except MABORT.

Description:

NOTE: For a more complete description of this utility, consult your Installation and Systems Management guide.

xxFRMT allows you to define drives and to format and partition disk drives. This utility is hardware dependent and will support up to four drives on one or two controllers. This utility is menu driven.

When you run xxFRMT, you select a drive with a letter and a select number character: 'F' or 'F0' selects floppy drive 0, or the lowest floppy drive select as defined for the system's disk controller; 'F1' for the second floppy drive; 'W' or 'W0 for the first Winchster drive; and 'W1,' 'W2', or 'W3' for the other possible system Winchester drives. Some systems may allow more floppies, and other systems may not allow up to four Winchester drives, but these six drives are the PDOS Winchester standard.

xxFRMT outputs a header message for the system, the names of the various controllers defined for that system, and the current P$PARM table entries with their controller numbers and select bytes. It then enters the Select Drive Menu.

```
x>xxFRMT
68K xxx Format Drive Utility
  Possible Disk Controllers in System are:
    Controller #1 is a xxx
    Controller #2 is a xxx
  Drives currently defined in system are:
    F0 is controller #1, drive select byte $00
    F1 is undefined.
    W0 is controller #1, drive select byte $00
    W1 is undefined.
    W2 is undefined.
    W3 is undefined.
 Select Menu: W,W0,W1,W2,W3=Winch;
              F,F0,F1=Floppy; Q=Quit
   Select Drive: _
```

If you select either a floppy drive or a Winchester drive that is already defined, xxFRMT directly enters the Drive Command Menu. If you are installing a new Winchester drive which is currently undefined, then you must enter the controller number and drive select jumpering (0-3). The Drive Command Menu tells you which drive you are currently dealing with and has the following commands:

```
 Select Menu: W,W0,W1,W2,W3=Winch; F,F0,F1=Floppy; Q=Quit
   Select Drive: W0
 W0 Main Menu: 1)Parm 2)BadT 3)Form 4)Veri 5) Part 6)Writ P)Togl Q)Quit
   Command: [CR]

Winchester Drive 0 Menu:
 1)Display/Alter Drive Parameters.    5)Disp/Alt PDOS Disk Partitions.
 2)Display/Alter Bad Track List.      6)Write out Header info to disk.
 3)Format tracks.                     P)Toggle Unit 2.
 4)Verify tracks                      Q)Quit & Select another Drive.
 W0 Main Menu: 1)Parm 2)BadT 3)Form 4)Veri 5) Part 6)Writ P)Togl Q)Quit
   Command: _
```

Continued on next page. . .

(7.34 xxFRMT - DISK HARDWARE FORMAT continued)

When dealing wih a floppy drive, the display/alter commands do not allow you to alter the drive parameters, the bad track table, or the disk partitions, and you do not write out the header information to a floppy disk. To exit to PDOS, you must first return to the Select Drive Menu with the Q) command.

Following is a brief description of the Drive Command Menu commands:

1)Display/Alter Drive Parameters

This option is used to reconfigure your drive. It allows you to D)isplay the currently defined drive parameters, A)lter them, R)ead them in from a file, or Q)uit and exit to the Select Drive Menu. The parameters that can be displayed/altered include the following:

number of heads on drive,
number of cylinders on drive,
physical blocks per track,
physical bytes per physical block,
shipping cylinder,
step rate,
reduced write current cylinder,
write precompensate cylinder.

2)Display/Alter Bad Track List

The Display/Alter Bad Track Menu allows you to D)isplay the currently defined bad tracks on the drive (if any), add or delete tracks, C)lear the bad track table, get a H)elp message, or Q)uit and exit to the Drive Command Menu.

3)Format Drive/Tracks

The Format Drive/Tracks option allows you to select the sector interleave and the physical tracks to format (defaults are provided). It then verifies that you want to format the drive and performs the format.

[CTRL-C] will abort the format. The track just formatted is printed on the screen. If there are errors, you can select either R)etry, Y)es (add the track to the bad track list), or N)o (ignore the error and go on).

(7.34 xxFRMT - DISK HARDWARE FORMAT continued)

4)Verify Tracks

The Verify Tracks option reads every sector on each track specified. [CTRL-C] will abort the verification. The track just verified is printed on the screen. If there are errors, you can select either R)etry, Y)es (add the track to the bad track list), or N)o (ignore the error and go on).

5)Display/Alter Disk Partitions

The Display/Alter Disk Partitions Menu allows you to D)isplay the currently defined disk partitions, A)lter them, R)ecalculate them from the current values, or Q)uit and exit to the Drive Command Menu.

6)Write Header Information to Drive

The Write Header Information to Drive Menu allows you to write the information to the drive header, abort the command and return to the Drive Command Menu, or write the drive information to a file. After assigning the correct parameters to a drive, entering the bad tracks, formatting it, and partitioning it into PDOS disk numbers, you still need to write this information to the drive's header. This information must reside on the disk for the BOOT ROMs and PDOS to assemble and use it.

P)Toggle Unit 2

The Toggle Unit 2 option allows you to print or send your output to some other device that is spooled to unit 2.

Q)Select Another Drive

If you are working with a floppy drive, the Q)uit command simply returns you to the Drive Select Menu. If you are working with a Winchester, it asks you whether or not to write the new drive data block down to low parameter RAM then returns you to the Drive Select Menu.

7.35 xxLDGO - LOAD AND/OR GO TO A NEW SYSTEM

Name: xxLDGO (where xx is the system ID)
Function: Load into memory and/or execute new system
Format: >xxLDGO
>xxLDGO {<load address>}{,<filename>}

Restrictions: xxLDGO will replace your current PDOS operating system and execute a new system,, terminating all tasks.

Description:

xxLDGO is used to load and execute new PDOS systems.

The <load address> is the location in memory where the program is to be located. A default is assumed for the hardware system.

The <filename> is the name of your system file. If a filename is not given, xxLDGO will look for a PDOS system in your task space. xxLDGO will only load a file in which the PDOS ID characters are found. After xxLDGO has loaded your new system, it will jump to the load address and begin execution.

xxLDGO allows you to try a new version of PDOS without modifying your boot image. Consult your Installation and Systems Management guide for details.

```
x>xxLDGO ,xxDOS
    DOS File Loaded: xxDOS
Found PDOS at address $0000AEB6
          DOS size is $000070CB
```

To make your new system into a disk boot, you need to follow the instructions for the MMKBT utility.

7.36 xxPARK - PARK DRIVES FOR SHIPPING

Name: xxPARK (where xx is the system ID)
Function: Flush buffers and park drives for shipping
Format: >xxPARK

Restrictions: Its use is system-dependent.

Description:

xxPARK parks the heads of drives that must be parked before shipping. On some systems, it is ineffective.

This utility also flushes the disk buffer so that disk data integrity is insured if disk buffering is enabled. It is only necessary to flush buffers with some controllers. Consult your Installation and Systems Management guide for specific information.

CHAPTER 8

BIOS, UARTs, DISK DSRs

All PDOS hardware dependence is confined to three modules, namely: 1) xxBIOS, which contains CPU-related parameters such as cold startup code, exception vector table, exception vector setup, DIP switches, memory mapper, clock acknowledgment, etc.; 2) xxBIOSU, which has all terminal I/O routines interfacing to various UARTs; and 3) xxBIOSW, which has the read and write logical sector routines. Another file, xxPARM:SR, is closely associated with the BIOS, is included when assembling the three BIOS modules, and defines various hardware addresses, offsets, and low parameter RAM locations used by the BIOS.

1. MBIOS - PDOS BASIC I/O SYSTEM

The PDOS BIOS module (MBIOS) is composed of the user BIOS module (xxBIOS:SR) and a common PDOS BIOS module (MBIOS:SR). The user BIOS module is composed of the task startup table (R$TASK) and various routines called by the PDOS common BIOS module and the PDOS kernel. These routines are optional and are only included when needed.

The user BIOS module is organized as follows:

B$STRT - Cold start entry address & constants
B$SRAM - System RAM address
R$TASK - Task startup table

B$CPU - Set CPU dependent parameters
B$RAM - Fix top of RAM
B$RSW - Read system switches

B$ACK - Acknowledge clock interrupt
B$LED - Blink LED & adjust clock
B$MAP - Load system map constant
B$SAV - Save hardware registers
B$RES - Restore hardware registers

BINTB - Interrupt vector table

SCRNTB - BASIC screen table

Text from the generic BIOS file MBIOS:SR is then included at the end of this user BIOS module.

1.1 - xxBIOS:SR - USER BIOS MODULE

The user BIOS module (xxBIOS:SR) consists of tables and routines specific to the system hardware. The following is an annotated boiler plate of a user BIOS module.

```
        TTL     xxBIOS:SR - 68K xxBIOS
*       xxBIOS:SR       11/17/86
************************************************************************
*                                                                      *
*       xx    xx  xx    xx  xx    xx  xx    xx  xx    xx  xx    xx     *
*        xx  xx    xx  xx    xx  xx    xx  xx    xx  xx    xx  xx      *
*         xxxx      xxxx      xxxx      xxxx      xxxx      xxxx       *
*          xx        xx        xx        xx        xx        xx        *
*         xxxx      xxxx      xxxx      xxxx      xxxx      xxxx       *
*        xx  xx    xx  xx    xx  xx    xx  xx    xx  xx    xx  xx      *
*       xx    xx  xx    xx  xx    xx  xx    xx  xx    xx  xx    xx     *
*                                                                      *
*               BBBBBBBB   IIIIII    0000000    SSSSSS                 *
*               BB     BB    II     00     00  SS                      *
*               BB     BB    II    00       00 SS                      *
*               BBBBBBBB     II    00       00  SSSSSS                 *
*               BB     BB    II    00       00       SS                *
*               BB     BB    II     00     00        SS                *
*               BBBBBBBB   IIIIII    0000000   SSSSSSS                 *
*                                                                      *
*=**********************************************************************
*=      REVISION SCHEDULE MODULE: xxBIOS
*=
xxBIOS IDNT     3.2     BIOS                                    IDNT label appears in QLINK map
*=
*=**********************************************************************
*
        IFUDF   RF      :RF     EQU 0           ;RUN MODULE FLAG           RF = Run Module flag
        IFUDF   TPS     :TPS    EQU 100         ;TICS/SECOND              TPS = System tics per second
        IFUDF   CLKADJ  :CLKADJ EQU 0           ;CLOCK ADJUST          CLKADJ = Clock adjustment factor
*
        OPT     ARS,CRE                                         MASM options for short absolute
        SECTION 14                                                references and cross reference
        PAGE
                                                                BIOSs are SECTION 14 code
```

(1.1 - xxBIOS:SR - USER BIOS MODULE continued)

```
*************************************************
*       RUN MODULE SECTION
*
        IFNE    RF                                          Define Task Startup Table External
        XREF    R$TASK,S$PROM                                 for Run Module assembly
        DC.L    SYZ.+S$SRAM     ;SUPERVISOR STACK POINTER   Also add EPROM 68000 startup vector
        DC.L    BSTRT           ;STARTUP VECTOR
        ENDC
*
*************************************************
*       PDOS ENTRY POINT
*
        XDEF    B$STRT          ;BIOS STARTUP ENTRY POINT   B$STRT = PDOS cold start entry addr
        XREF    B$SRAM          ;ADDRESS OF SYRAM POINTER   B$SRAM = System RAM variable
        XREF    S$SRAM          ;SYSTEM RAM                 S$SRAM = System RAM (Defined at link time)
*                                                             PDID = 'PDOS'
B$STRT  BRA.L   BSTRT           ;BOOT EPROM START             SYID = System identification
        DC.L    PDID            ;PDOS BOOT IDENTIFICATION
        DC.W    SYID            ;SYSTEM ID
B.SRAM  DC.L    S$SRAM          ;SYRAM ADDRESS
        XREF    U.1ADR,U.1TYP                               U.xADR = UART base address
        XREF    U.2ADR,U.2TYP                               U.xTYP = UART type
*
*************************************************
*       TASK STARTUP TABLE (NON-RUN MODULE)
*
        IFEQ    RF                                          R$TASK = Task Startup Table
        XDEF    R$TASK

*
R$TASK  DC.B    1,U.1TYP,BIBR,%0000     ;PORT #1
        DC.L    U.1ADR
        DC.B    2,U.2TYP,BIBR,%0000     ;PORT #2
        DC.L    U.2ADR
        DC.W    0                       ;END-OF-TABLE
*
*       TASK #0
*
        DC.B    64              ;PRIORITY                   Task priority (1-255)
        DC.B    TT              ;TASK TIME                  Task time slice
        DC.L    0               ;DSEG SIZE                  RAM size (0=use all)
        DC.W    0               ;MAP                        Mapper constant
        DC.L    *-*             ;PSEG START (0=MBEGN)       Task entry address (0=Monitor)
        DC.W    1               ;PORT #                     Task port #

        <Insert other startup tasks here>

        DC.W    0               ;END OF TABLE
        ENDC
```

(1.1 - xxBIOS:SR - USER BIOS MODULE continued)

```
BMESO1 DC.B    $0A,$0D,'xxBIOS ',$DATE,0
       EVEN
```

BMESO1 = BIOS startup message

```
**************************************************
*      CPU DEPENDENT PARAMETERS
*
B.PTMSK        EQU     $2500          ;PORT DISABLE INTERRUPT MASK
SYID   EQU     'xx'                   ;SYSTEM ID WORD
*
**************************************************
*      CPU DEPENDENT SETUP ROUTINES
*
B$CPU  EQU     *               ;CPU SETUP
```

B.PTMSK = Disable all port interrupts

The B$CPU routine initializes the system. This may include the system clock, memory mapper, interrupts, controllers, or any other CPU dependent parameters.

```
       RTS
*
**************************************************
*      FIX TOP OF RAM
*
B$RAM  EQU     *               ;RAM FIX
```

B$RAM In: (A2) = Top of RAM
(A4) = BIOS table
(A5) = SYRAM
(A7) = (Top of RAM)-4 (RTS)

The B$RAM routine is called after memory has been sized. It is here that the top of memory (A7) can be adjusted for special buffers.

```
       RTS
*
**************************************************
*      READ SWITCHES
*
B$RSW  EQU     *               ;READ SWITCHES
```

B$RSW In: D4.L = SYRAM (B.BAS) bit map base (=0)
D5.W = Baud rate (-1=none)
D6.L = B.VEC=vector base register (=0)
D7.L = $00/ASF.B/FLG$.B/SDK$.B
(A3) = Interrupt vector table (BINTB)
(A4) = BIOS table (B$BIOS)
(A6) = Start of tasking memory
(A7) = End of tasking memory

The B$RSW routine is called just before entering the PDOS kernel. It is here that system switches can be read and the initial baud rate (D5.W), auto-start flag (ASF.B), or system disk (SDK$.B) adjusted.

```
       RTS
       PAGE
```

(1.1 - xxBIOS:SR - USER BIOS MODULE continued)

```
*************************************************
*      ACKNOWLEDGE CLOCK INTERRUPT
*
B$ACK  EQU    *                ;ACKNOWLEDGE CLOCK
```

B$ACK = Acknowledge clock interrupt

The B$ACK routine is called by the PDOS kernel every clock interrupt. B$ACK is to acknowledge the interrupt. Address register A5 points to SYRAM.

```
       RTS
*
*************************************************
*      BLINK LED & ADJUST CLOCK
*
B$LED  EQU    *                ;BLINK LED
```

B$LED = Blink LED & adjust clock

The B$LED routine is called by the PDOS kernel once every second. If there is a system LED, it is toggled to indicate that PDOS is up and tasking properly. The BCLK variable is also examined to determine if the system clock needs fine tuning.

```
       MOVE.L  B_CLK(A0),D0    ;ADJUST CLOCK?
         BEQ.S @0002           ;N
       ADD.L   D0,BCLK.(A5)    ;Y, ADJUST COUNT, CARRY?
         BCC.S @0002           ;N
       ADDQ.W  #1,FCNT.(A5)    ;Y, UP COUNTER
*
@0002  RTS                     ;RETURN
*
*************************************************
*      LOAD SYSTEM MAP CONSTANT
*
B$MAP  EQU    *                ;LOAD MAP CONSTANT
```

B$MAP = Load system map constant

The B$MAP routine is called by the PDOS kernel every time a task is scheduled or when a task's memory is referenced by another task using system primitives. Data register D0.W has the map constant, address register A5 points to SYRAM, and address register A0 points to the BIOS table.

```
       RTS
```

(1.1 - xxBIOS:SR - USER BIOS MODULE continued)

```
*
****************************************************
*       SAVE 68881 REGISTERS ON USER STACK
*
        OPT     P=68020,P=68881,OLD
*
B$SAV   FSAVE   -(A1)
        FMOVEM.X FP0-FP7,-(A1)  ;SAVE 68881 REGISTERS FP0-FP1
        FMOVE.L FPCR/FPSR,-(A1) ;SAVE STATUS REGISTER
        RTS                     ;RETURN
```

The B$SAV routine is called by the PDOS kernel before every task context switch if the task save flag (SVF$) is set. Address register A1 contains the User Stack Pointer (USP) which is saved on the Supervisor Stack immediately on return. Address register A5 points to SYRAM, and register A0 points to the BIOS table.

```
*
***************************************************
*       RESTORE FROM STACK
*
B$RES   FMOVE.L (A1)+,FPCR/FPSR
        FMOVEM.X (A1)+,FP0-FP7
        FRESTORE (A1)+
        RTS                     ;RETURN
        PAGE
```

The B$RES routine is called by the PDOS kernel after every task context switch if the task save flag (SVF$) is set. Address register A1 contains the User Stack Pointer (USP) which is restored to the 68000 USP register immediately on return. Address register A5 points to SYRAM, and register A0 points to the BIOS table.

(1.1 - xxBIOS:SR - USER BIOS MODULE continued)

```
**************************************************
*       INTERRUPT VECTOR GENERATION:
*
*       The MC68000 interrupt vectors are built during PDOS
*       initialization according to the 'BINTB' table.  Each
*       exception vector entry consists of a word address
*       for the vector and a long word, B$BIOS relative entry
*       for the exception processing routine.
*
*       SOFT ENTRIES INTO THE PDOS KERNEL ARE DEFINED AS FOLLOWS:
*
*               K1$STRT = PDOS INITIALIZATION
*
*               K2$CHRI = CHARACTER IN PROCESSOR
*                       1) DISABLE INTERRUPTS
*                       2) DO-D7/AO-A6 ON SYSTEM STACK
*                       3) AO.L=UART BASE ADDRESS
*                       4) DO.B=CHARACTER
*                       5) 'MOVEA.L B$SRAM,A5'
*                       6) 'BRA.L K2$CHRI'
*                       7) ROUTINE WILL EXIT WITH CONTEXT SWITCH
*
*               K1$CLKI = SYSTEM CLOCK PROCESSOR
*                       1) ONLY SR & PC ON SUPERVISOR STACK
*                          (CLOCK PROCESSOR WILL STACK REGISTERS.)
*                       2) 'BRA.L K1$CLKI'
*
*               K1$SERR = SYSTEM ERROR PROCESSOR
*                       1) SUPERVISOR STACK SHOULD LOOK AS FOLLOWS:
*                               (A7) = DC.L  (MESSAGE)
*                                      DC.W  LADR,R/W,I/N,CODE
*                                      DC.L  ACCESS ADDRESS
*                                      DC.W  INSTRUCTION REGISTER
*                                      DC.W  STATUS REGISTER
*                                      DC.L  PROGRAM COUNTER
*                       2) 'BRA.L K1$SERR'
        PAGE
```

(1.1 - xxBIOS:SR - USER BIOS MODULE continued)

```
**************************************************
*       xxBIOS INTERRUPT STRUCTURE
*
BINTB  EQU     *                ;INTERRUPT TABLE
       DC.W    $007C            ;INTERRUPT LEVEL 7 PROCESSOR
       DC.L    BINT7-B$BIOS
```

All system-dependent exception vectors are built from three-word entries of the following format:

```
       DC.W    <address>
       DC.L    <routine>-B$BIOS ;ADDITIONAL VECTORS
       ....
       DC.W    0                ;END-OF-TABLE
*
**************************************************
*       INTERRUPT LEVEL 7 PROCESSOR
*
BINT7  EQU     *                ;INTERRUPT LEVEL 7
```

Interrupt 7 in BIOS!

Interrupt level 7 must be processed in the system BIOS. This may be for parity errors, abort switches, or system memory refresh.

```
       RTE                      ;RETURN FROM INTERRUPT
       PAGE


**********************************************************
*       SCREEN COMMAND TABLE
*
SFLG   EQU     0                ;SCREEN TABLE FLAG
*
SCRNTB DC.B    <code>,<letter>
       ....
       DC.B    0
```

If SFLG=0, Include SCRNTB
If SFLG=1, Use MBIOS screen table

Finally, the common MBIOS:SR module is included to complete the BIOS module.

```
       INCLUDE MBIOS:SR         ;INCLUDE COMMON BIOS MODULE
       END
```

1.2 - MBIOS:SR - COMMON BIOS MODULE

The common BIOS module (MBIOS:SR) is included at the end of the user BIOS module. It has many default equates that also can be adjusted at assembly time. The BIOS configuration table (B$BIOS) drives the PDOS system and is pointed to by the first long word of SYRAM.

In addition, MBIOS:SR contains some user-alterable, cold start-up code which initializes the hardware, sizes memory, sets up the RAM disk, and loads registers, then branches to the generic PDOS kernel startup entry point.

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                    68020 PDOS Assembler 06-Nov-86
PAGE: 1          15:03 30-Nov-86    FILE: MBIOS:SR,WDISK #4

*=      MBIOS:SR        10/31/86
*******************************************************************************
*                                                                             *
*   PPPPPP  DDDDDD    0000    SSSS          BBBBBB  IIII   0000    SSSS       *
*   PP   PP DD   DD  00  00  SS  SS         BB   BB  II   00  00  SS  SS      *
*   PP   PP DD    DD 00    00 SS            BB   BB  II  00    00 SS          *
*   PPPPPP  DD    DD 00    00  SSS          BBBBBB   II  00    00  SSS        *
*   PP      DD    DD 00    00     SS        BB   BB  II  00    00     SS      *
*   PP      DD   DD  00  00  SS  SS         BB   BB  II   00  00  SS  SS      *
*   PP      DDDDDD    0000    SSSS          BBBBBB  IIII   0000    SSSS       *
*                                                                             *
*******************************************************************************
*                                                                             *
*       Copyright 1983-1986 Eyring Research Institute, Inc.                   *
*                           1450 West 820 North                               *
*                           Provo, UT USA                                     *
*                           All Rights Reserved.                              *
*                                                                             *
*=*****************************************************************************
*=      REVISION SCHEDULE MODULE: MBIOS
*=
*=      10/31/86 3.10   B_SYS with system parameters
*=                      8 megabyte BUS ERR sizing fixed
*=      10/27/86 3.9    B.SLV added for default file directory level
*=                      B.WND added for window control character
*=                      TT defaults to 1
*=      10/08/86 3.8    B$AXRT replaced by B_CMD
*=      09/29/86 3.7    B.RDA references fixed
*=                      Port # prompt when windowing
*=                      B.CMD for command delimiter ('.')
*=                      B.EXT for file extension (':')
*=                      B.LEV for file level (';')
*=                      B.DSK for file disk ('/')
*=                      B.WC1 for character wild card ('*')
*=                      B.WC2 for field wild card ('@')
*=      08/07/86 3.6    IRD anded with RZ
*=                      A6 placed on 2k boundary
*=                      K1$STRT entry w/B.BAS=Memory base, B.VEC=Vector base
*=      05/02/86 3.5    B.TEV - Toggle event number
*=      03/27/86 3.4    B.BRK & B.CLR - break & clear buffer characters
*=                      B.SZ1 & B.SZ2 - default create task sizes
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                68020 PDOS Assembler 06-Nov-86
PAGE: 2        15:04 30-Nov-86  FILE: MBIOS:SR,WDISK #4

*=                  B.TTM - default task time
*=  02/04/86 3.3    B_SAV,B_RES added
*=  09/05/85 3.2    B$AXRT added
*=  08/23/85 3.1    Release 3.0a -- DGF
*=                  -- Improve EQU on B.ADD..
*=                  -- Fix ANSI terminal support
*=  07/25/85 3.0    Release 3.0 B.RGT CHANGED TO ^L-- dgf
*=  06/03/85 2.15   R.xx values eliminated
*=  05/14/85 2.14   BIBR -> BR
*=                  Prompt altered to handle SDS$=-1
*=  04/19/85 2.13   Eliminated B.SYMSK
*=  04/16/85 2.12   BINTC eliminated
*=  04/05/84 2.11   Check for 68010
*=  02/28/85 2.10   EPROM check on BUS error
*=                  BD68 Flags
*=                  B$MPT moved to MBIOS
*=  01/17/84 2.9    DIVU.W #2048,D1 >> LSR.L #11,D1
*=                  JSR B$IRD(A4) >> BSR.S B$IRD
*=  01/08/84 2.8    HR = High RAM Address
*=  12/20/84 2.7    B_TPS added to BIOS table
*=                  B_PSC,B_CLS & B$PSC,B$CLS
*=                  B_IRD & B$IRD for init RAM disk
*=  07/16/84 2.6    B_IVC.W, B$MPT FOR PDOS MONITOR PROMPT
*=  04/12/84 2.5    CHECK FOR RAM DISK INITIALIZED
*=                  CLEAR RAM DISK
*=  04/04/84 2.4    B.PTMSK/B.SYMSK CHECK
*=  01/20/84 2.3    BINTB W/O
*=  01/05/84 2.2    PINT FORCES CONTEXT SWITCH
*=                  IRMDK - INITIALIZE RAM DISK
*=  10/18/83 2.1    READ SWITCHES FOR BAUD RATE
*=                  RAM DISK=8
*=  09/15/83        SYSRAM
*=  06/29/83        SYID = SYSTEM ID
*=  06/23/83        B_CLK = CLOCK ADJUST FACTOR
*=
*=***************************************************************************
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                         68020 PDOS Assembler 06-Nov-86
PAGE: 3       15:04 30-Nov-86            FILE: MBIOS:SR,WDISK #4

                                  **************************************************
                                  *       EXTERNAL DEFINITIONS
                                  *
           00000004               XDEF    B_SID         ;SYSTEM ID WORD
           00000006               XDEF    B_TPS         ;SYSTEM TICS/SECOND
           00000008               XDEF    B_CLK         ;CLOCK ADJUST FACTOR
           0000000C               XDEF    B_TEV         ;EVENTS 112-115
           0000001C               XDEF    B_DAF         ;SYSTEM FLAGS
           0000001E               XDEF    B_URT         ;UART DSR ROUTINES
           00000028               XDEF    B_CPC         ;CSC$/PSC$ = CLEAR & POSITION CODES
           0000002C               XDEF    B_MSZ         ;MAIL ARRAY SIZE
           0000002E               XDEF    B_PDM         ;PDOS MONITOR PROMPT
           00000034               XDEF    B_RDK         ;RAM DISK PARAMETERS
           0000003E               XDEF    B_ACK         ;SYSTEM CLOCK ACKNOWLEDGE
           00000042               XDEF    B_CTB         ;SYSTEM CREATE TASK
           00000046               XDEF    B_KTB         ;SYSTEM KILL TASK
           0000004A               XDEF    B_LED         ;SYSTEM LED
           0000004E               XDEF    B_MAP         ;SYSTEM SCHEDULE TASK (LOAD MAP)
           00000052               XDEF    B_PRT         ;SYSTEM PROTECT
           00000056               XDEF    B_PSC         ;POSITION CURSOR
           0000005A               XDEF    B_CLS         ;CLEAR SCREEN
           0000005E               XDEF    B_IRD         ;INIT RAM DISK
           00000062               XDEF    B_DIT         ;DISK INITIALIZATION    (EXTERNAL ABSOLUTE)
           00000066               XDEF    B_DOF         ;DISK MOTORS OFF        (EXTERNAL ABSOLUTE)
           0000006A               XDEF    B_RSE         ;READ SECTOR            (EXTERNAL ABSOLUTE)
           0000006E               XDEF    B_WSE         ;WRITE SECTOR           (EXTERNAL ABSOLUTE)
           00000072               XDEF    B_SFN         ;AUTO START FILE NAME
           00000074               XDEF    B_SCT         ;BASIC SCREEN TABLE
           00000076               XDEF    B_MES         ;BIOS MESSAGE
           00000078               XDEF    B_SAV         ;SAVE ON STACK
           0000007C               XDEF    B_RES         ;RESTORE FROM STACK
           00000080               XDEF    B_CMD         ;MONTIOR COMMAND
           000000A0               XDEF    B_SYS         ;SYSTEM PARAMETERS
                                  *
           00000000               XDEF    B.PTMSK       ;DISABLE PORT INTERRUPT MASK
           00000000               XDEF    B$BIOS        ;BIOS CONFIGURATION TABLE
           00000020               XDEF    B.RDE         ;# OF RAM DISK DIRECTORY ENTRIES
           00000008               XDEF    B.RDU         ;RAM DISK UNIT #
           000000FF               XDEF    B.RDZ         ;RAM DISK SIZE
           00000064               XDEF    B.TPS         ;SYSTEM TICS/SECOND
           00000001               XDEF    B.ADD         ;RECALL LINE CHARACTER
           00000008               XDEF    B.LFT         ;MOVE CURSOR LEFT CHARACTER
           0000000C               XDEF    B.RGT         ;MOVE CURSOR RIGHT CHARACTER
           00000004               XDEF    B.DRT         ;DELETE RIGHT CHARACTER
           0000007F               XDEF    B.DLT         ;DELETE LEFT CHARACTER
           00000003               XDEF    B.BRK         ;PDOS BREAK CHARACTER
           00000018               XDEF    B.CLR         ;PDOS CLEAR BUFFER CHARACTER
           00000018               XDEF    B.WND         ;PDOS WINDOW CONTROL CHARACTER
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                        68020 PDOS Assembler 06-Nov-86
PAGE: 4        15:04 30-Nov-86          FILE: MBIOS:SR,WDISK #4

        00000004                XDEF    B.SZ1           ;DEFAULT @ TASK SIZE
        00000020                XDEF    B.SZ2           ;DEFAULT CT TASK SIZE
        00000001                XDEF    B.TTM           ;DEFAULT CT TASK TIME
        00000040                XDEF    B.TEV           ;TOGGLE EVENT #
        0000002E                XDEF    B.CMD           ;COMMAND DELIMITER ('.')
        0000003A                XDEF    B.EXT           ;FILE EXTENSION (':')
        0000003B                XDEF    B.LEV           ;FILE LEVEL (';')
        0000002F                XDEF    B.DSK           ;FILE DISK ('/')
        0000002A                XDEF    B.WC1           ;CHARACTER WILD CARD ('*')
        00000040                XDEF    B.WC2           ;FIELD WILD CARD ('@')
        00000001                XDEF    B.SLV           ;FILE DIRECTORY LEVEL
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                      68020 PDOS Assembler 06-Nov-86
PAGE: 5         15:04 30-Nov-86       FILE: MBIOS:SR,WDISK #4

                                 *************************************************
                                 *       EXTERNAL REFERENCES
                                 *
                                 *       LINKS TO KERNEL MODULES
                                 *
                                         XREF.1  K1$STRT       ;PDOS KERNEL ENTRY POINT
                                         XREF.1  K1$CLKI       ;CLOCK ENTRY
                                         XREF.1  K1$SERR       ;EXTERNAL SYSTEM ERROR
                                         XREF.1  K2$PINT       ;PORT INTERRUPT SERVICE ROUTINE
                                         XREF.1  K2$CHRI       ;EXTERNAL CHARACTER IN
                                         XREF.1  K2$CHAR       ;INSERT CHARACTER
                                 *
                                 *       LINKS TO SYRAM & SYSTEM CONSTANTS
                                 *
                                         XREF.1  SYZ.,MBZ.
                                         XREF.1  FCNT.,BCLK.
                                         XREF.1  BFLG.,F681.
                                         XREF.1  MAPB.,NMB.
                                         XREF.1  PORT.,NPS.
                                         XREF.1  NCP.,IOUT.
                                         XREF.1  TQUE.,TLST.
                                         XREF.1  NTB.,TBZ.
                                         XREF.1  TMTF.,TMBF.
                                         XREF.1  NTM.,TMZ.
                                         XREF.1  TMSP.,NTP.
                                         XREF.1  DEVT.,NEV.
                                         XREF.1  XCHB.,NCB.
                                         XREF.1  XFSL.,NFS.
                                         XREF.1  TBE$
                                 *
                                 *       LINKS TO UART MODULE
                                 *
                                         XREF.1  U$1DSR,U$2DSR
                                         XREF.1  U$3DSR,U$4DSR
                                 *
                                 *       LINKS TO R/W SECTOR MODULE
                                 *
                                         XREF.1  W$XWSE,W$XRSE
                                         XREF.1  W$XDIT,W$XDOF
                                 *
                                 ********************************************************
                                 *
                       0000A55A XPID     EQU     $A55A         ;PDOS ID
                       50444F53 PDID     EQU     'PDOS'
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                  68020 PDOS Assembler 06-Nov-86
PAGE: 6        15:04 30-Nov-86    FILE: MBIOS:SR,WDISK #4

                          *********************************************************
                          *       SYSTEM DEFAULT PARAMETERS
                          *********************************************************
                          *
                                 IFUDF   TPS     :TPS     EQU 100         ;TICS/SECOND
                                 IFUDF   BPS     :BPS     EQU 256         ;BYTES/SECTOR
                                 IFUDF   AS      :AS      EQU 1           ;AUTO START FLAG
                                 IFUDF   MZ      :MZ      EQU 256         ;MAIL ARRAY SIZE
                                 IFUDF   TT      :TT      EQU 1           ;TASK TIME
                                 IFUDF   HR      :HR      EQU 0           ;HIGHEST MEMORY ADR
                          *
                                 IFUDF   B.ADD   :B.ADD   EQU 'A'-'@'     ;RECALL LAST LINE
                                 IFUDF   B.LFT   :B.LFT   EQU 'H'-'@'     ;MOVE LEFT
                                 IFUDF   B.RGT   :B.RGT   EQU 'L'-'@'     ;MOVE RIGHT
                                 IFUDF   B.DRT   :B.DRT   EQU 'D'-'@'     ;DELETE RIGHT
                                 IFUDF   B.DLT   :B.DLT   EQU $7F         ;DELETE LEFT
                                 IFUDF   B.BRK   :B.BRK   EQU 'C'-'@'     ;PDOS BREAK
                                 IFUDF   B.CLR   :B.CLR   EQU 'X'-'@'     ;CLEAR BUFFER
                                 IFUDF   B.WND   :B.WND   EQU 'X'-'@'     ;PDOS WINDOW CONTROL CHARACTER
                                 IFUDF   B.CMD   :B.CMD   EQU '.'         ;COMMAND DELIMITER
                                 IFUDF   B.EXT   :B.EXT   EQU ':'         ;FILE EXTENSION
                                 IFUDF   B.LEV   :B.LEV   EQU ';'         ;FILE LEVEL
                                 IFUDF   B.DSK   :B.DSK   EQU '/'         ;FILE DISK
                                 IFUDF   B.WC1   :B.WC1   EQU '*'         ;CHARACTER WILD CARD
                                 IFUDF   B.WC2   :B.WC2   EQU '@'         ;FIELD WILD CARD
                                 IFUDF   SD      :SD      EQU 0           ;DEFAULT DISK #
                                 IFUDF   SF      :SF      EQU 0           ;SYSTEM FLAGS
                                 IFUDF   BR      :BR      EQU 0           ;INITIAL BAUD RATE
                                 IFUDF   LV      :LV      EQU 1           ;FILE DIRECTORY LEVEL
                                 IFUDF   CPSC    :CPSC    EQU $AA009B3D   ;CLEAR & POSITION CODES
                                 IFUDF   EV112   :EV112   EQU TPS/5       ;EVENT 112
                                 IFUDF   EV113   :EV113   EQU 1           ;EVENT 113
                                 IFUDF   EV114   :EV114   EQU 10          ;EVENT 114
                                 IFUDF   EV115   :EV115   EQU 20          ;EVENT 115
                                 IFUDF   B.SZ1   :B.SZ1   EQU 4           ;DEFAULT @ TASK SIZE
                                 IFUDF   B.SZ2   :B.SZ2   EQU 32          ;DEFAULT CT TASK SIZE
                                 IFUDF   B.TEV   :B.TEV   EQU 64          ;DEFAULT TOGGLE EVENT #
                                 IFUDF   B.BAS   :B.BAS   EQU 0           ;DEFAULT MEMORY MAP BASE
                                 IFUDF   B.VEC   :B.VEC   EQU 0           ;DEFAULT VECTOR BASE
                          *
                 00000064 B.TPS  EQU     TPS               ;EXTERNAL TICS/SECOND
                 00000001 B.TTM  EQU     TT                ;TASK TIME SLICE
                 00000001 B.SLV  EQU     LV                ;FILE DIRECTORY LEVEL
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                          68020 PDOS Assembler 06-Nov-86
PAGE: 7          15:04 30-Nov-86          FILE: MBIOS:SR,WDISK #4

                                   ********************************************************
                                   *       RAM DISK PARAMETERS
                                   ********************************************************
                                   *
                                           IFUDF   RU :RU EQU 8         ;RAM DISK UNIT
                                           IFUDF   RZ :RZ EQU 255       ;RAM DISK SIZE
                                           IFUDF   RE :RE EQU (RZ/8)!7+1   ;# OF DIRECTORY ENTRIES
                                           IFUDF   RA :RA EQU 0         ;RAM DISK ADDRESS
                                   *
                   00000008 B.RDU          EQU     RU               ;RAM DISK UNIT
                   000000FF B.RDZ          EQU     RZ               ;RAM DISK SIZE
                   00000020 B.RDE          EQU     RE               ;# OF DIRECTORY ENTRIES
                                   *
0/00000000:                                IFNE    (RA>0)&(RA<128)
                                           XREF    B.RDA            ;EXTERNAL RAM DISK (0<RA<128)
                                           ENDC
0/00000000:                                IFNE    (RA<=0)!(RA>127)
           00000000                        XDEF    B.RDA            ;RAM DISK ADDRESS
                   00000000 B.RDA          EQU     RA               ;RAM DISK ADDRESS (RA>127)
                                           ENDC
                                   *
                                   ********************************************************
                                   *       DEFAULT FLAGS
                                   ********************************************************
                                   *
                                           IFUDF   FBA     :FBA EQU 0     ;BASIC
                                           IFUDF   FDR     :FDR EQU 0     ;DIRECTORY FLAG
                                   *
                                   ********************************************************
                                   *       MBIOS SUBROUTINE FLAGS
                                   ********************************************************
                                   *
                                           IFUDF   IRD     :IRD EQU 1*RZ  ;RAM DISK INITIALIZATION
                                           IFUDF   ANS     :ANS EQU 1     ;ANSI 3.64 PSC/CSC
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                    68020 PDOS Assembler 06-Nov-86
PAGE: 8        15:04 30-Nov-86      FILE: MBIOS:SR,WDISK #4

                        ************************************************************
                        *       MBIOS SUBROUTINES
                        ************************************************************
                        *
                                IFDEF   B$ACK   :IBACK  SET $60000000+(B$ACK-B$BIOS-B_ACK-2)&$FFFF
              4E750000          IFUDF   B$ACK   :IBACK  SET $4E750000
                                IFDEF   B$CTB   :IBCTB  SET $60000000+(B$CTB-B$BIOS-B_CTB-2)&$FFFF
              4E750000          IFUDF   B$CTB   :IBCTB  SET $4E750000
                                IFDEF   B$KTB   :IBKTB  SET $60000000+(B$KTB-B$BIOS-B_KTB-2)&$FFFF
              4E750000          IFUDF   B$KTB   :IBKTB  SET $4E750000
                                IFDEF   B$LED   :IBLED  SET $60000000+(B$LED-B$BIOS-B_LED-2)&$FFFF
              4E750000          IFUDF   B$LED   :IBLED  SET $4E750000
                                IFDEF   B$MAP   :IBMAP  SET $60000000+(B$MAP-B$BIOS-B_MAP-2)&$FFFF
              4E750000          IFUDF   B$MAP   :IBMAP  SET $4E750000
                                IFDEF   B$PRT   :IBPRT  SET $60000000+(B$PRT-B$BIOS-B_PRT-2)&$FFFF
              4E750000          IFUDF   B$PRT   :IBPRT  SET $4E750000
              600001F4          IFDEF   B$PSC   :IBPSC  SET $60000000+(B$PSC-B$BIOS-B_PSC-2)&$FFFF
                                IFUDF   B$PSC   :IBPSC  SET $4E750000
              60000236          IFDEF   B$CLS   :IBCLS  SET $60000000+(B$CLS-B$BIOS-B_CLS-2)&$FFFF
                                IFUDF   B$CLS   :IBCLS  SET $4E750000
              6000015C          IFDEF   B$IRD   :IBIRD  SET $60000000+(B$IRD-B$BIOS-B_IRD-2)&$FFFF
                                IFUDF   B$IRD   :IBIRD  SET $4E750000
                                IFDEF   B$SAV   :IBSAV  SET $60000000+(B$SAV-B$BIOS-B_SAV-2)&$FFFF
              4E750000          IFUDF   B$SAV   :IBSAV  SET $4E750000
                                IFDEF   B$RES   :IBRES  SET $60000000+(B$RES-B$BIOS-B_RES-2)&$FFFF
              4E750000          IFUDF   B$RES   :IBRES  SET $4E750000
                                IFDEF   B$CMD   :IBCMD  SET $60000000+(B$CMD-B$BIOS-B_CMD-2)&$FFFF
              4E750000          IFUDF   B$CMD   :IBCMD  SET $4E750000
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                       68020 PDOS Assembler 06-Nov-86
PAGE: 9        15:04 30-Nov-86         FILE: MBIOS:SR,WDISK #4

                                ****************************************************
                                *      BIOS CONFIGURATION TABLE
                                *
                                *      *NOTE:  PRESERVE THE ORDER OF THIS TABLE!
                                *
0/00000000:00000000             B$BIOS DC.L     R$TASK-B$BIOS   ;TASK STARTUP TABLES    (EXTERNAL ABSOLUTE)
                       00000004 B_SID  EQU      *-B$BIOS        ;SYSTEM ID WORD
0/00000004:0000                        DC.W     SYID
                       00000006 B_TPS  EQU      *-B$BIOS        ;SYSTEM TICS/SECOND
0/00000006:0064                        DC.W     TPS
                       00000008 B_CLK  EQU      *-B$BIOS        ;CLOCK ADJUST FACTOR
0/00000008:00000000                    DC.L     CLKADJ
                       0000000C B_TEV  EQU      *-B$BIOS        ;EVENTS 112-115
0/0000000C:00000014                    DC.L     EV112
0/00000010:00000001                    DC.L     EV113
0/00000014:0000000A                    DC.L     EV114
0/00000018:00000014                    DC.L     EV115
                       0000001C B_DAF  EQU      *-B$BIOS        ;SYSTEM FLAGS
0/0000001C:00                          DC.B     0
0/0000001D:01                          DC.B     AS              ;AUTO-START FLAG
0/0000001E:00                          DC.B     SF              ;SYSTEM FLAGS
0/0000001F:00                          DC.B     SD              ;SYSTEM DISK
                       0000001E B_URT  EQU      *-B$BIOS-2      ;UART DSR ROUTINES
0/00000020:****                        DC.W     U$1DSR-B$BIOS
0/00000022:****                        DC.W     U$2DSR-B$BIOS
0/00000024:****                        DC.W     U$3DSR-B$BIOS
0/00000026:****                        DC.W     U$4DSR-B$BIOS
                       00000028 B_CPC  EQU      *-B$BIOS        ;CSC$/PSC$ = CLEAR & POSITION CODES
0/00000028:AA009B3D                    DC.L     CPSC
                       0000002C B_MSZ  EQU      *-B$BIOS        ;MAIL ARRAY SIZE
0/0000002C:0100                        DC.W     MZ
                       0000002E B_PDM  EQU      *-B$BIOS        ;PDOS MONITOR PROMPT
0/0000002E:600001F6                    BRA.L    B$MPT
                       00000032        EQU      *-B$BIOS        ;SPARE
0/00000032:0000                        DC.W     0
                       00000034 B_RDK  EQU      *-B$BIOS        ;RAM DISK PARAMETERS
0/00000034:0008                        DC.W     B.RDU           ;RAM DISK UNIT
0/00000036:00FF                        DC.W     B.RDZ           ;RAM DISK SIZE
0/00000038:00000000                    DC.L     B.RDA           ;RAM DISK ADDRESS
0/0000003C:0020                        DC.W     B.RDE           ;# OF DIRECTORY ENTRIES
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                              68020 PDOS Assembler 06-Nov-86
PAGE: 10        15:04 30-Nov-86               FILE: MBIOS:SR,WDISK #4

                                    **************************************************
                                    *       BIOS CONFIGURATION TABLE (continued)
                                    *
                         0000003E B_ACK   EQU     *-B$BIOS        ;SYSTEM CLOCK ACKNOWLEDGE
0/0000003E:4E750000               DC.L    IBACK
                         00000042 B_CTB   EQU     *-B$BIOS        ;SYSTEM CREATE TASK
0/00000042:4E750000               DC.L    IBCTB
                         00000046 B_KTB   EQU     *-B$BIOS        ;SYSTEM KILL TASK
0/00000046:4E750000               DC.L    IBKTB
                         0000004A B_LED   EQU     *-B$BIOS        ;SYSTEM LED
0/0000004A:4E750000               DC.L    IBLED
                         0000004E B_MAP   EQU     *-B$BIOS        ;SYSTEM SCHEDULE TASK (LOAD MAP)
0/0000004E:4E750000               DC.L    IBMAP
                         00000052 B_PRT   EQU     *-B$BIOS        ;SYSTEM PROTECT
0/00000052:4E750000               DC.L    IBPRT
                         00000056 B_PSC   EQU     *-B$BIOS        ;POSITION CURSOR
0/00000056:600001F4               DC.L    IBPSC
                         0000005A B_CLS   EQU     *-B$BIOS        ;CLEAR SCREEN
0/0000005A:60000236               DC.L    IBCLS
                         0000005E B_IRD   EQU     *-B$BIOS        ;INIT RAM DISK
0/0000005E:6000015C               DC.L    IBIRD
                         00000062 B_DIT   EQU     *-B$BIOS        ;DISK INITIALIZATION    (EXTERNAL ABSOLUTE)
0/00000062:********               DC.L    W$XDIT-B$BIOS
                         00000066 B_DOF   EQU     *-B$BIOS        ;DISK MOTORS OFF        (EXTERNAL ABSOLUTE)
0/00000066:********               DC.L    W$XDOF-B$BIOS
                         0000006A B_RSE   EQU     *-B$BIOS        ;READ SECTOR            (EXTERNAL ABSOLUTE)
0/0000006A:********               DC.L    W$XRSE-B$BIOS
                         0000006E B_WSE   EQU     *-B$BIOS        ;WRITE SECTOR           (EXTERNAL ABSOLUTE)
0/0000006E:********               DC.L    W$XWSE-B$BIOS
                         00000072 B_SFN   EQU     *-B$BIOS        ;AUTO START FILE NAME
0/00000072:00CE                   DC.W    STRTFL-B$BIOS
                         00000074 B_SCT   EQU     *-B$BIOS        ;BASIC SCREEN TABLE
0/00000074:02A6                   DC.W    SCRNTB-B$BIOS
                         00000076 B_MES   EQU     *-B$BIOS        ;BIOS MESSAGE
0/00000076:0000                   DC.W    BMES01-B$BIOS
                         00000078 B_SAV   EQU     *-B$BIOS        ;SAVE ON STACK
0/00000078:4E750000               DC.L    IBSAV
                         0000007C B_RES   EQU     *-B$BIOS        ;RESTORE FROM STACK
0/0000007C:4E750000               DC.L    IBRES
                         00000080 B_CMD   EQU     *-B$BIOS        ;MONITOR COMMANDS
0/00000080:4E750000               DC.L    IBCMD
0/00000084:000E0000               DCB.B   $A0+B$BIOS-*,0  ;SPARES
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                          68020 PDOS Assembler 06-Nov-86
PAGE: 11        15:04 30-Nov-86           FILE: MBIOS:SR,WDISK #4

                                 *************************************************
                                 *       PDOS SYSTEM PARAMETERS
                                 *
                       000000A0 B_SYS   EQU     *-B$BIOS        ;SYSTEM PARAMETERS
0/000000A0:****                         DC.W    TBE$            ;$A0 = TASK CONTROL BLOCK SIZE
0/000000A2:****                         DC.W    MAPB.           ;$A2 = SYSTEM MEMORY BIT MAP
0/000000A4:****                         DC.W    NMB.            ;$A4 = MAP SIZE
0/000000A6:****                         DC.W    PORT.           ;$A6 = INPUT CHARACTER BUFFERS
0/000000A8:****                         DC.W    NPS.            ;$A8 = # OF I/O PORTS
0/000000AA:****                         DC.W    NCP.            ;$AA = # OF CHARACTERS/PORT
0/000000AC:****                         DC.W    IOUT.           ;$AC = OUTPUT CHARACTER BUFFERS
0/000000AE:****                         DC.W    TQUE.           ;$AE = TASK QUEUE
0/000000B0:****                         DC.W    TLST.           ;$B0 = TASK LIST
0/000000B2:****                         DC.W    NTB.            ;$B2 = MAX # OF TASKS
0/000000B4:****                         DC.W    TBZ.            ;$B4 = TASK LIST ENTRY SIZE
0/000000B6:****                         DC.W    TMTF.           ;$B6 = TO/FROM INDEX TABLE
0/000000B8:****                         DC.W    TMBF.           ;$B8 = TASK MESSAGE BUFFERS
0/000000BA:****                         DC.W    NTM.            ;$BA = # TASK MESSAGES
0/000000BC:****                         DC.W    TMZ.            ;$BC = TASK MESSAGE SIZE
0/000000BE:****                         DC.W    TMSP.           ;$BE = TASK MESSAGE POINTERS
0/000000C0:****                         DC.W    NTP.            ;$C0 = # OF TASK MESSAGE POINTERS
0/000000C2:****                         DC.W    DEVT.           ;$C2 = DELAY EVENT LIST
0/000000C4:****                         DC.W    NEV.            ;$C4 = # OF DELAY EVENTS
0/000000C6:****                         DC.W    XCHB.           ;$C6 = CHANNEL BUFFERS
0/000000C8:****                         DC.W    NCB.            ;$C8 = # OF BUFFERS
0/000000CA:****                         DC.W    XFSL.           ;$CA = FILE SLOTS
0/000000CC:****                         DC.W    NFS.            ;$CC = # OF FILE SLOTS
                                 *
                                 *************************************************
                                 *       MISCELLANEOUS STRING CONSTANTS
                                 *
0/000000CE:5359245354525400 STRTFL  DC.B    'SY$STRT',0     ;FILE START NAME
                                 *
0/000000D6:53592444534B00   RDNM    DC.B    'SY$DSK',0      ;RAM DISK NAME
0/000000DD:        0/000000DE       EVEN
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                   68020 PDOS Assembler 06-Nov-86
PAGE: 12      15:04 30-Nov-86      FILE: MBIOS:SR,WDISK #4

                            **************************************************
                            *       PDOS STARTUP CODE
                            **************************************************
                            *
                            *       1. Set map registers
                            *       2. Start system clock
                            *       3. Return parameters
                            *
0/000000DE:2A7AFF20         BSTRT   MOVEA.L B.SRAM(PC),A5   ;POINT TO SYRAM
0/000000E2:23CD00000000             MOVE.L  A5,B$SRAM       ;SAVE IN RAM
0/000000E8:4FED****                 LEA.L   SYZ.(A5),A7     ;SET SUPERVISOR STACK POINTER
0/000000EC:2C4F                     MOVEA.L A7,A6           ;POINT TO START OF TASKING MEMORY
0/000000EE:41FA0026                 LEA.L   BSTR04(PC),A0
0/000000F2:23C800000010             MOVE.L  A0,16           ;SET ILLEGAL ERROR
0/000000F8:B1F900000010             CMPA.L  16,A0           ;CHANGED?
0/000000FE:670C                       BEQ.S BSTR02          ;Y
0/00000100:227900000010             MOVEA.L 16,A1           ;N
0/00000106:32FC4EF9                 MOVE.W  #$4EF9,(A1)+    ;OUTPUT 'JMP.L <A0>'
0/0000010A:2288                     MOVE.L  A0,(A1)
                            *
0/0000010C:426D****         BSTR02  CLR.W   F681.(A5)       ;DEFAULT TO 68000
0/00000110:42C0                     DC.W    $42C0           ;'MOVE.W CCR,D0'
0/00000112:546D****                 ADDQ.W  #2,F681.(A5)    ;MUST BE 68010 IF THIS IS EXECUTED
                            *
0/00000116:200E             BSTR04  MOVE.L  A6,D0           ;PUT TASKING MEMORY ON 2K BOUNDARY
0/00000118:0680000007FF             ADDI.L  #2048-1,D0
0/0000011E:0280FFFFF800             ANDI.L  #-2048,D0
0/00000124:2C40                     MOVEA.L D0,A6
0/00000126:2E4E                     MOVEA.L A6,A7           ;RESTORE STACK POINTER
                                    IFDEF   B$CPU : BSR.L B$CPU     ;DO HARDWARE DEPENDENT SETUPS
0/00000128:49FAFED6                 LEA.L   B$BIOS(PC),A4   ;POINT TO BIOS
0/0000012C:                         IFNE    HR
                                    LEA.L   HR,A2           ;POINT TO TOP OF RAM
                                    ENDC    HR
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                                68020 PDOS Assembler 06-Nov-86
PAGE: 13        15:04 30-Nov-86                 FILE: MBIOS:SR,WDISK #4

                                   **************************************************
                                   *       SIZE MEMORY
                                   *
0/0000012C:                                IFEQ    HR
0/0000012C:203C********                    MOVE.L  #MBZ.,D0        ;GET # OF MAP 2K BLOCKS
0/00000132:220E                            MOVE.L  A6,D1
0/00000134:E089                            LSR.L   #8,D1           ;D1/2048
0/00000136:E689                            LSR.L   #3,D1
0/00000138:9081                            SUB.L   D1,D0
0/0000013A:244E                            MOVEA.L A6,A2           ;POINT TO BEGINNING OF TASKING MEMORY
0/0000013C:41FA0028                        LEA.L   BSTR10(PC),A0
0/00000140:23C800000008                    MOVE.L  A0,8            ;SET NEW BUS ERROR
0/00000146:B1F900000008                    CMPA.L  8,A0            ;CHANGED?
0/0000014C:670C                              BEQ.S BSTR06          ;Y
0/0000014E:227900000008                    MOVEA.L 8,A1            ;N
0/00000154:32FC4EF9                        MOVE.W  #$4EF9,(A1)+    ;OUTPUT 'JMP.L <A0>'
0/00000158:2288                            MOVE.L  A0,(A1)
                                   *
0/0000015A:4A52                    BSTR06  TST.W   (A2)            ;BUS ERROR?
0/0000015C:5380                            SUBQ.L  #1,D0           ;N, TRY MORE?
0/0000015E:6706                              BEQ.S BSTR08          ;Y
0/00000160:D4FC0800                        ADDA.W  #$800,A2        ;N, MOVE TO NEXT
0/00000164:60F4                            BRA.S   BSTR06
                                   *
                        0/00000166 BSTR08  EQU     *
                                           ENDC    HR
                                   *
                                   **************************************************
                                   *       BUS ERROR OR D0=0
                                   *       START SYSTEM CLOCK
                                   *
0/00000166:2E4A                    BSTR10  MOVEA.L A2,A7           ;A7 = TOP OF RAM
                                           IFDEF   B$RAM : BSR.L B$RAM     ;FIX TOP OFF RAM
0/00000168:224D                            MOVEA.L A5,A1           ;POINT TO PDOS SYSTEM RAM
0/0000016A:22CC                            MOVE.L  A4,(A1)+        ;BIOS. = A4
0/0000016C:9EEC002C                        SUBA.W  B_MSZ(A4),A7    ;MAKE ROOM FOR MAIL ARRAY
0/00000170:22CF                            MOVE.L  A7,(A1)+        ;MAIL. = MAIL ARRAY POINTER
0/00000172:22EC0034                        MOVE.L  B_RDK(A4),(A1)+ ;RDKN. = RAM DISK UNIT/SIZE
0/00000176:4291                            CLR.L   (A1)            ;RDKA. = RAM DISK ADDRESS
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                        68020 PDOS Assembler 06-Nov-86
PAGE: 14        15:04 30-Nov-86         FILE: MBIOS:SR,WDISK #4

                                **********************************************
                                *       FIX RAM DISK
                                *
0/00000178:3E2C0036                     MOVE.W  B_RDK+2(A4),D7  ;RAM DISK?
0/0000017C:671E                           BEQ.S BSTR14          ;N
0/0000017E:3007                         MOVE.W  D7,D0           ;Y
0/00000180:C0FC0100                     MULU.W  #BPS,D0         ;GET SIZE
0/00000184:222C0038                     MOVE.L  B_RDK+4(A4),D1  ;ADDRESS DEFINED?
0/00000188:6604                           BNE.S BSTR12          ;Y
0/0000018A:9FC0                         SUBA.L  D0,A7           ;N, MAKE ROOM FOR RAM DISK
0/0000018C:220F                         MOVE.L  A7,D1
                                *
0/0000018E:2281                 BSTR12  MOVE.L  D1,(A1)         ;STORE RAM DISK ADDRESS
0/00000190:                             IFNE    IRD
0/00000190:2041                         MOVEA.L D1,A0           ;GET ADDRESS
0/00000192:0C68A55A001C                 CMPI.W  #XPID,28(A0)    ;ALREADY INITIALIZED?
0/00000198:6702                           BEQ.S BSTR14          ;Y
0/0000019A:6120                         BSR.S B$IRD             ;N, INIT RAM DISK (D1.L = DISK ADR)
                                        ENDC
                                *
                                *       SETUP FINAL REGISTERS & ENTER PDOS
                                *
                                *       D4.L = MEMORY BIT MAP BASE ADDRESS
                                *       D5.W = BAUD RATE (-1=NONE)
                                *       D6.L = EXCEPTION VECTOR BASE ADDRESS
                                *       D7.L = $00 / AUTO.B / FLG$.B / SDK$.B
                                *       (A3) = VECTOR TABLE
                                *       (A4) = BIOS TABLE
                                *       (A6) = START OF TASKING MEMORY
                                *       (A7) = END OF TASKING MEMORY
                                *
0/0000019C:7AFF                 BSTR14  MOVEQ.L #-1,D5          ;USE START TABLE FOR BAUD RATES
0/0000019E:2E2C001C                     MOVE.L  B_DAF(A4),D7    ;SET D7.L
0/000001A2:47FAFE5C                     LEA.L   BINTB(PC),A3    ;POINT TO INTERRUPT TABLE
0/000001A6:3B7C0000****                 MOVE.W  #FBA<<8+FDR,BFLG.(A5)   ;BFLG.B, DFLG.B
                                        IFDEF   B$RSW : BSR.L B$RSW     ;GET SWITCHES
0/000001AC:283C00000000                 MOVE.L  #B.BAS,D4       ;MEMORY BASE
0/000001B2:2C3C00000000                 MOVE.L  #B.VEC,D6       ;VECTOR BASE
0/000001B8:6000****                     BRA.L   K1$STRT         ;GOOOOO!!!!
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                           68020 PDOS Assembler 06-Nov-86
PAGE: 15        15:04 30-Nov-86            FILE: MBIOS:SR,WDISK #4

                                  **************************************************
                                  *       INITIALIZE RAM DISK
                                  *
                                  *       D1.L = RAM DISK ADDRESS
                                  *       D7.W = NPS
                                  *       (A4) = BIOS
                                  *
0/000001BC:                               IFNE     IRD
0/000001BC:2041                   B$IRD   MOVEA.L D1,A0              ;POINT TO RAM DISK
0/000001BE:3007                           MOVE.W  D7,D0              ;GET SIZE
0/000001C0:C0FC0100                       MULU.W  #BPS,D0            ;GET # OF BYTES
                                  *
0/000001C4:4298                   @IR02   CLR.L   (A0)+              ;CLEAR RAM DISK
0/000001C6:5980                           SUBQ.L  #4,D0              ;DONE?
0/000001C8:6EFA                             BGT.S @IR02              ;N
0/000001CA:2441                           MOVEA.L D1,A2              ;Y, POINT TO RAM DISK
0/000001CC:3C2C003C                       MOVE.W  B_RDK+8(A4),D6     ;D6.W = NDE
0/000001D0:41EA0018                       LEA.L   24(A2),A0          ;POINT TO HEADER INFORMATION
0/000001D4:43FAFF00                       LEA.L   RDNM(PC),A1        ;POINT TO NAME
                                  *
0/000001D8:14D9                   @IR04   MOVE.B  (A1)+,(A2)+        ;MOVE IN NAME, DONE?
0/000001DA:66FC                             BNE.S @IR04              ;N
0/000001DC:30C6                           MOVE.W  D6,(A0)+           ;Y, 24 = SAVE NDE
0/000001DE:30C7                           MOVE.W  D7,(A0)+           ;   26 = SAVE NPS
0/000001E0:20BCA55A0000                   MOVE.L  #XPID<<16,(A0)     ;28/30 = ID/SIDES-DENSITY
0/000001E6:5648                           ADDQ.W  #3,A0              ;POINT TO 31
0/000001E8:343CFF00                       MOVE.W  #-32*8,D2          ;224 MAP BYTES IN HEADER
0/000001EC:3606                           MOVE.W  D6,D3              ;CALCULATE # OF DIRECTORY SECTORS
0/000001EE:E64E                           LSR.W   #3,D6              ;8 ENTRIES/SECTOR
0/000001F0:EB0B                           LSL.B   #8-3,D3            ;PARTIAL SECTOR?
0/000001F2:6702                             BEQ.S @IR06              ;N
0/000001F4:5246                           ADDQ.W  #1,D6              ;Y, ALLOCATE WHOLE SECTOR
                                  *
0/000001F6:5246                   @IR06   ADDQ.W  #1,D6              ;BIT MAP SECTORS
0/000001F8:06420800                       ADDI.W  #BPS*8,D2
0/000001FC:B447                           CMP.W   D7,D2              ;ENOUGH SECTORS?
0/000001FE:65F6                             BLO.S @IR06              ;N
0/00000200:7601                           MOVEQ.L #1,D3              ;Y, GET MASK
0/00000202:7200                           MOVEQ.L #0,D1              ;START WITH SECTOR ZERO
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                        68020 PDOS Assembler 06-Nov-86
PAGE: 16        15:04 30-Nov-86         FILE: MBIOS:SR,WDISK #4

                               **************************************************
                               *       CREATE RAM DISK DIRECTORY
                               *
0/00000204:E21B                @IR08   ROR.B    #1,D3          ;MOVE TO NEXT SECTOR, WRAP AROUND?
0/00000206:6404                          BCC.S @IR10           ;N
0/00000208:5248                        ADDQ.W  #1,A0           ;Y, MOVE TO NEXT BYTE
0/0000020A:50D0                        ST.B    (A0)            ;ALLOCATE 8 SECTORS
                               *
0/0000020C:B246                @IR10   CMP.W   D6,D1           ;Y, ALLOCATE SECTOR?
0/0000020E:6502                          BLO.S @IR12           ;N, CLEAR BUFFER
0/00000210:B710                        EOR.B   D3,(A0)         ;Y
                               *
0/00000212:5241                @IR12   ADDQ.W  #1,D1           ;MOVE TO NEXT SECTOR
0/00000214:B247                        CMP.W   D7,D1           ;DONE?
0/00000216:65EC                          BLO.S @IR08           ;N
0/00000218:5248                        ADDQ.W  #1,A0           ;Y, USE LAST WORD
                               *
0/0000021A:2008                @IR14   MOVE.L  A0,D0           ;GET ADDRESS
0/0000021C:4A00                        TST.B   D0              ;DONE (256 BYTE BOUNDARY)?
0/0000021E:6704                          BEQ.S @IR16           ;Y
0/00000220:50D8                        ST.B    (A0)+           ;N, FINISH ALLOCATING SECTOR
0/00000222:60F6                        BRA.S   @IR14
                               *
0/00000224:4E75                @IR16   RTS                     ;RETURN
                                       ENDC
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                          68020 PDOS Assembler 06-Nov-86
PAGE: 17        15:04 30-Nov-86           FILE: MBIOS:SR,WDISK #4

                               **************************************************
                               *       PDOS MONITOR PROMPT
                               *
                                       IFUDF B$MPT :B$MPT EQU *
   0/00000226:                         IFEQ  B$MPT-*
                                       XREF.1  SDS$,SLV$
                               *
                               *
   0/00000226:242E****                 MOVE.L  SDS$(A6),D2       ;GET DISK(S)
   0/0000022A:7604                     MOVEQ.L #4,D3             ;GET COUNTER
   0/0000022C:A088                     XPCL                      ;OUTPUT CRLF
   0/0000022E:7000                     MOVEQ.L #0,D0             ;1ST DELIMITER=0
                               *
   0/00000230:7200             @0004   MOVEQ.L #0,D1
   0/00000232:1202                     MOVE.B  D2,D1             ;GET DISK #
   0/00000234:5202                     ADDQ.B  #1,D2             ;OK?
   0/00000236:6708                       BEQ.S @0006             ;N
   0/00000238:A086                     XPCC                      ;Y, OUTPUT 1ST CHARACTER
   0/0000023A:702C                     MOVEQ.L #',',D0           ;CHANGE TO COMMA
   0/0000023C:A050                     XCBD
   0/0000023E:A08A                     XPLC                      ;OUTPUT DISK #
                               *
   0/00000240:E08A             @0006   LSR.L   #8,D2             ;ADJUST D2
   0/00000242:5303                     SUBQ.B  #1,D3             ;DONE?
   0/00000244:6EEA                       BGT.S @0004             ;N
   0/00000246:703E                     MOVEQ.L #'>',D0           ;Y
   0/00000248:A086                     XPCC
   0/0000024A:4E75                     RTS                       ;RETURN
                                       ENDC
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                      68020 PDOS Assembler 06-Nov-86
PAGE: 18       15:04 30-Nov-86        FILE: MBIOS:SR,WDISK #4

                               *************************************************
                               *      ANSI 3.64 (VT100) - POSITION CURSOR
                               *
                               *      IN:     D1.B = ROW POSITION
                               *              D2.B = COLUMN POSITION
                               *              (A3) = CBO$(A6)
                               *      OUT:      SR = .NE.
                               *
                               *      ANSI MODE = <esc>[xxx;yyyH
                               *
0/0000024C:                           IFNE    ANS
0/0000024C:36FC9BDB            B$PSC  MOVE.W  #$9B00+$80+'[',(A3)+
0/00000250:7000                       MOVEQ.L #0,D0           ;CONVERT TO 32  BIT UNSIGNED
0/00000252:1001                       MOVE.B  D1,D0           ;GET ROW POSITION
0/00000254:6114                       BSR.S   @0002           ;ROUTINE TO COMPUTE OCTAL POSITIONING
0/00000256:16FC00BB                   MOVE.B  #$80+';',(A3)+
0/0000025A:7000                       MOVEQ.L #0,D0           ;CONVERT TO 32 BIT UNSIGNED
0/0000025C:1002                       MOVE.B  D2,D0           ;GET COLUMN POSITION
0/0000025E:610A                       BSR.S   @0002           ;ROUTINE TO COMPUTE OCTAL POSITIONING
0/00000260:16FC00C8                   MOVE.B  #$80+'H',(A3)+
0/00000264:421B                       CLR.B   (A3)+
0/00000266:4267                       CLR.W   -(SP)
0/00000268:4E77                       RTR                     ;RETURN .NE.
                               *
0/0000026A:5280                @0002  ADDQ.L  #1,D0           ;BIAS ROW/COL BY 1
0/0000026C:80FC0064                   DIVU.W  #100,D0         ;GET NUMBER OF 100S
0/00000270:4A40                       TST.W   D0
0/00000272:6706                         BEQ.S @0004           ;NONE
0/00000274:060000B0                   ADDI.B  #$80+'0',D0     ;OUTPUT NUMBER
0/00000278:16C0                       MOVE.B  D0,(A3)+
                               *
0/0000027A:4840                @0004  SWAP    D0              ;GET 10'S
0/0000027C:48C0                       EXT.L   D0
0/0000027E:80FC000A                   DIVU.W  #10,D0
0/00000282:060000B0                   ADDI.B  #$80+'0',D0     ;OUTPUT 10'S
0/00000286:16C0                       MOVE.B  D0,(A3)+
0/00000288:4840                       SWAP    D0
0/0000028A:060000B0                   ADDI.B  #$80+'0',D0     ;OUTPUT 1'S
0/0000028E:16C0                       MOVE.B  D0,(A3)+
0/00000290:4E75                       RTS                     ;RETURN TO CALLER
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                         68020 PDOS Assembler 06-Nov-86
PAGE: 19        15:04 30-Nov-86          FILE: MBIOS:SR,WDISK #4

                             *************************************************
                             *       ANSI 3.64 (VT100)  - CLEAR SCREEN
                             *
                             *       ANSI 3.64 MODE = <esc>[2J<esc>[H
                             *
                             *       IN:
                             *       OUT:       SR = .NE.
                             *
0/00000292:45FA000A          B$CLS   LEA.L   ANSCLR(PC),A2   ;POINT TO CLEAR SCREEN SEQUENCE
                             *
0/00000296:16DA              @0002   MOVE.B  (A2)+,(A3)+     ;OUTPUT, DONE?
0/00000298:66FC                        BNE.S @0002           ;N
0/0000029A:4267                      CLR.W   -(A7)           ;Y, PUSH .NE.
0/0000029C:4E77                      RTR                     ;RETURN
                             *
0/0000029E:9BDBB2CA          ANSCLR  DC.B    $9B,$80+'[',$80+'2',$80+'J'     ;ANSI CLEAR DISPLAY
0/000002A2:9BDBC800                  DC.B    $9B,$80+'[',$80+'H',0           ;ANSI MOVE CURSOR HOME
0/000002A6:                          EVEN
                                     ENDC
                             *
                             *********************************************************
                             *       SCREEN COMMAND TABLE
                             *
                                     IFUDF SCRNTB :SCRNTB EQU *
0/000002A6:                          IFEQ SCRNTB-*
                             *
0/000002A6:0B55                      DC.B    $0B,'U'         ;U = UP
0/000002A8:0A44                      DC.B    $0A,'D'         ;D = DOWN
0/000002AA:0C52                      DC.B    $0C,'R'         ;R = RIGHT
0/000002AC:084C                      DC.B    $08,'L'         ;L = LEFT
0/000002AE:0D42                      DC.B    $0D,'B'         ;B = BEGINNING
0/000002B0:1E48                      DC.B    $1E,'H'         ;H = HOME
0/000002B2:D953                      DC.B    $D9,'S'         ;S = CLEAR TO END OF SCREEN
0/000002B4:D445                      DC.B    $D4,'E'         ;E = CLEAR TO END OF LINE
0/000002B6:A757                      DC.B    $A7,'W'         ;W = RESET WRITE PROTECT
0/000002B8:A650                      DC.B    $A6,'P'         ;P = SET WRITE PROTECT
0/000002BA:A928                      DC.B    $A9,'('         ;( = START WRITE PROTECT
0/000002BC:A829                      DC.B    $A8,')'         ;) = END WRITE PROTECT
0/000002BE:AB5A                      DC.B    $AB,'Z'         ;Z = CLEAR UNPROTECTED
0/000002C0:094E                      DC.B    $09,'N'         ;N = SKIP TO NEXT FIELD
0/000002C2:0000                      DC.W    0               ;END-OF-TABLE
                                     ENDC
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                       68020 PDOS Assembler 06-Nov-86
PAGE: 20       15:04 30-Nov-86         FILE: MBIOS:SR,WDISK #4

                                *******************************************************************************
                                *       DISPLAY ASSEMBLY RESULTS
                                *
0/000002C4:                             IFNE    MZ<>(MZ/2)*2
                                         FAIL   ERROR >> 'MZ' MUST BE EVEN!
                                        ENDC
                                *
                     00000064           PRINT   '>> Tics/second      TPS = ',TPS
                     00000001           PRINT   '>> Auto start flag   AS = ',AS
                     00000100           PRINT   '>> Mail array        MZ = ',MZ
                     00000000           PRINT   '>> System disk       SD = ',SD
                     00000000           PRINT   '>> System flags      SF = ',SF
                     00000000           PRINT   '>> Baud rate         BR = ',BR
                     00000000           PRINT   '>> BASIC flag       FBA = ',FBA
                     00000000           PRINT   '>> Directory flag   FDR = ',FDR
                                *
0/000002C4:                             IFEQ    RZ
                                         PRINT  '>> No RAM disk'
                                        ENDC    RZ
0/000002C4:                             IFNE    RZ
                     00000008            PRINT  '>> RAM disk unit     RU = ',RU
                     000000FF            PRINT  '>> RAM disk size     RZ = ',RZ
0/000002C4:                              IFEQ   RA
                                          PRINT '>> RAM disk allocated from top of memory'
                                         ENDC   RA
0/000002C4:                              IFNE   RA
                                          PRINT '>> RAM disk addr     RA = $',$RA
                                         ENDC   RA
                                        ENDC    RZ
                                *
0/000002C4:                             IFEQ    HR
                                         PRINT  '>> Size memory'
                                        ENDC    HR
0/000002C4:                             IFNE    HR
                                         PRINT  '>> High memory address = $',HR
                                        ENDC    HR
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                        68020 PDOS Assembler 06-Nov-86
PAGE: 21        15:04 30-Nov-86         FILE: MBIOS:SR,WDISK #4

                                        IFDEF   B$ACK   : PRINT '>> User system routine B$ACK included.'
                                        IFDEF   B$CPU   : PRINT '>> User system routine B$CPU included.'
                                        IFDEF   B$RAM   : PRINT '>> User system routine B$RAM included.'
                                        IFDEF   B$RSW   : PRINT '>> User system routine B$RSW included.'
                                        IFDEF   B$CTB   : PRINT '>> User system routine B$CTB included.'
                                        IFDEF   B$KTB   : PRINT '>> User system routine B$KTB included.'
                                        IFDEF   B$LED   : PRINT '>> User system routine B$LED included.'
                                        IFDEF   B$MAP   : PRINT '>> User system routine B$MAP included.'
                                        IFDEF   B$PRT   : PRINT '>> User system routine B$PRT included.'
                                        IFDEF   B$PSC   : PRINT '>> PDOS system routine B$PSC included.'
                                        IFDEF   B$CLS   : PRINT '>> PDOS system routine B$CLS included.'
                                        IFDEF   B$IRD   : PRINT '>> PDOS system routine B$IRD included.'
                                        IFDEF   B$SAV   : PRINT '>> PDOS system routine B$SAV included.'
                                        IFDEF   B$RES   : PRINT '>> PDOS system routine B$RES included.'
   0/000002C4:                          IFNE    ANS
                                         PRINT  '>> ANSI 3.64 position and clear screen routine included.'
                                        ENDC
                                *
                                ********END OF FILE****************************************
   0/000002C4:                          END
```

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

68020 PDOS Assembler 06-Nov-86
PAGE: 22   15:04 30-Nov-86   FILE: MBIOS:SR,WDISK #4

DEFINED SYMBOLS:

ANS	E	00000001	ANSCLR		0/0000029E	AS	E	00000001
B$BIOS	D	0/00000000	B$CLS		0/00000292	B$IRD		0/000001BC
B$MPT	E	0/00000226	B$PSC		0/0000024C	B$SRAM	E	00000000
B.ADD	ED	00000001	B.BAS	E	00000000	B.BRK	ED	00000003
B.CLR	ED	00000018	B.CMD	ED	0000002E	B.DLT	ED	0000007F
B.DRT	ED	00000004	B.DSK	ED	0000002F	B.EXT	ED	0000003A
B.LEV	ED	0000003B	B.LFT	ED	00000008	B.PTMSK	ED	00000000
B.RDA	E	00000000	B.RDE	E	00000020	B.RDU	E	00000008
B.RDZ	E	000000FF	B.RGT	ED	0000000C	B.SLV	E	00000001
B.SRAM		0/00000000	B.SZ1	ED	00000004	B.SZ2	ED	00000020
B.TEV	ED	00000040	B.TPS	E	00000064	B.TTM	E	00000001
B.VEC	E	00000000	B.WC1	ED	0000002A	B.WC2	ED	00000040
B.WND	ED	00000018	BCLK.	XR	X/00000000	BFLG.	X	X/00000000
BINTB		0/00000000	BMES01		0/00000000	BPS	E	00000100
BR	E	00000000	BSTR02		0/0000010C	BSTR04		0/00000116
BSTR06		0/0000015A	BSTR08	E	0/00000166	BSTR10		0/00000166
BSTR12		0/0000018E	BSTR14		0/0000019C	BSTRT	R	0/000000DE
B_ACK	ED	0000003E	B_CLK	ED	00000008	B_CLS	ED	0000005A
B_CMD	ED	00000080	B_CPC	ED	00000028	B_CTB	ED	00000042
B_DAF	ED	0000001C	B_DIT	ED	00000062	B_DOF	ED	00000066
B_IRD	ED	0000005E	B_KTB	ED	00000046	B_LED	ED	0000004A
B_MAP	ED	0000004E	B_MES	ED	00000076	B_MSZ	ED	0000002C
B_PDM	ED	0000002E	B_PRT	ED	00000052	B_PSC	ED	00000056
B_RDK	ED	00000034	B_RES	ED	0000007C	B_RSE	ED	0000006A
B_SAV	ED	00000078	B_SCT	ED	00000074	B_SFN	ED	00000072
B_SID	ED	00000004	B_SYS	ED	000000A0	B_TEV	ED	0000000C
B_TPS	ED	00000006	B_URT	ED	0000001E	B_WSE	ED	0000006E
CLKADJ	E	00000000	CPSC	E	AA009B3D	DEVT.	X	X/00000000
EV112	E	00000014	EV113	E	00000001	EV114	E	0000000A
EV115	E	00000014	F681.	X	X/00000000	FBA	E	00000000
FCNT.	XR	X/00000000	FDR	E	00000000	HR	E	00000000
IBACK	S	4E750000	IBCLS	S	60000236	IBCMD	S	4E750000
IBCTB	S	4E750000	IBIRD	S	6000015C	IBKTB	S	4E750000
IBLED	S	4E750000	IBMAP	S	4E750000	IBPRT	S	4E750000
IBPSC	S	600001F4	IBRES	S	4E750000	IBSAV	S	4E750000
IOUT.	X	X/00000000	IRD	E	000000FF	K1$CLKI	XR	X/00000000
K1$SERR	XR	X/00000000	K1$STRT	X	X/00000000	K2$CHAR	XR	X/00000000
K2$CHRI	XR	X/00000000	K2$PINT	XR	X/00000000	LF	E	00000000
LV	E	00000001	MAPB.	X	X/00000000	MBZ.	X	X/00000000
MZ	E	00000100	NCB.	X	X/00000000	NCP.	X	X/00000000
NEV.	X	X/00000000	NFS.	X	X/00000000	NMB.	X	X/00000000
NPS.	X	X/00000000	NTB.	X	X/00000000	NTM.	X	X/00000000
NTP.	X	X/00000000	PDID	ER	50444F53	PORT.	X	X/00000000

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

68020 PDOS Assembler 06-Nov-86
PAGE: 23 15:04 30-Nov-86 FILE: MBIOS:SR,WDISK #4

R$TASK		0/00000000	RA	E	00000000	RDNM		0/000000D6
RE	E	00000020	RU	E	00000008	RZ	E	000000FF
SCRNTB	E	0/000002A6	SD	E	00000000	SDS$	X	X/00000000
SF	E	00000000	SLV$	XR	X/00000000	STRTFL		0/000000CE
SYID	E	00000000	SYZ.	X	X/00000000	TBE$	X	X/00000000
TBZ.	X	X/00000000	TLST.	X	X/00000000	TMBF.	X	X/00000000
TMSP.	X	X/00000000	TMTF.	X	X/00000000	TMZ.	X	X/00000000
TPS	E	00000064	TQUE.	X	X/00000000	TT	E	00000001
U$1DSR	X	X/00000000	U$2DSR	X	X/00000000	U$3DSR	X	X/00000000
U$4DSR	X	X/00000000	W$XDIT	X	X/00000000	W$XDOF	X	X/00000000
W$XRSE	X	X/00000000	W$XWSE	X	X/00000000	WF	E	00000000
XCHB.	X	X/00000000	XFSL.	X	X/00000000	XPID	E	0000A55A

EXTERNAL DEFINITIONS:

B$BIOS	D	0/00000000	B.ADD	ED	00000001	B.BRK	ED	00000003
B.CLR	ED	00000018	B.CMD	ED	0000002E	B.DLT	ED	0000007F
B.DRT	ED	00000004	B.DSK	ED	0000002F	B.EXT	ED	0000003A
B.LEV	ED	0000003B	B.LFT	ED	00000008	B.PTMSK	ED	00000000
B.RGT	ED	0000000C	B.SZ1	ED	00000004	B.SZ2	ED	00000020
B.TEV	ED	00000040	B.WC1	ED	0000002A	B.WC2	ED	00000040
B.WND	ED	00000018	B_ACK	ED	0000003E	B_CLK	ED	00000008
B_CLS	ED	0000005A	B_CMD	ED	00000080	B_CPC	ED	00000028
B_CTB	ED	00000042	B_DAF	ED	0000001C	B_DIT	ED	00000062
B_DOF	ED	00000066	B_IRD	ED	0000005E	B_KTB	ED	00000046
B_LED	ED	0000004A	B_MAP	ED	0000004E	B_MES	ED	00000076
B_MSZ	ED	0000002C	B_PDM	ED	0000002E	B_PRT	ED	00000052
B_PSC	ED	00000056	B_RDK	ED	00000034	B_RES	ED	0000007C
B_RSE	ED	0000006A	B_SAV	ED	00000078	B_SCT	ED	00000074
B_SFN	ED	00000072	B_SID	ED	00000004	B_SYS	ED	000000A0
B_TEV	ED	0000000C	B_TPS	ED	00000006	B_URT	ED	0000001E
B_WSE	ED	0000006E						

EXTERNAL REFERENCES:

BCLK.	XR	X/00000000	BFLG.	X	X/00000000	DEVT.	X	X/00000000
F681.	X	X/00000000	FCNT.	XR	X/00000000	IOUT.	X	X/00000000
K1$CLKI	XR	X/00000000	K1$SERR	XR	X/00000000	K1$STRT	X	X/00000000
K2$CHAR	XR	X/00000000	K2$CHRI	XR	X/00000000	K2$PINT	XR	X/00000000
MAPB.	X	X/00000000	MBZ.	X	X/00000000	NCB.	X	X/00000000
NCP.	X	X/00000000	NEV.	X	X/00000000	NFS.	X	X/00000000
NMB.	X	X/00000000	NPS.	X	X/00000000	NTB.	X	X/00000000
NTM.	X	X/00000000	NTP.	X	X/00000000	PORT.	X	X/00000000
SDS$	X	X/00000000	SLV$	XR	X/00000000	SYZ.	X	X/00000000
TBE$	X	X/00000000	TBZ.	X	X/00000000	TLST.	X	X/00000000
TMBF.	X	X/00000000	TMSP.	X	X/00000000	TMTF.	X	X/00000000
TMZ.	X	X/00000000	TQUE.	X	X/00000000	U$1DSR	X	X/00000000
U$2DSR	X	X/00000000	U$3DSR	X	X/00000000	U$4DSR	X	X/00000000
W$XDIT	X	X/00000000	W$XDOF	X	X/00000000	W$XRSE	X	X/00000000
W$XWSE	X	X/00000000	XCHB.	X	X/00000000	XFSL.	X	X/00000000

(1.2 - MBIOS:SR - COMMON BIOS MODULE continued)

```
                                        68020 PDOS Assembler 06-Nov-86
PAGE: 24        15:04 30-Nov-86         FILE: MBIOS:SR,WDISK #4


UNDEFINED SYMBOLS:

B$ACK     UR    00000000   B$CMD     UR    00000000   B$CPU     UR    00000000
B$CTB     UR    00000000   B$KTB     UR    00000000   B$LED     UR    00000000
B$MAP     UR    00000000   B$PRT     UR    00000000   B$RAM     UR    00000000
B$RES     UR    00000000   B$RSW     UR    00000000   B$SAV     UR    00000000

UNREFERENCED SYMBOLS:

BCLK.     XR  X/00000000   BSTRT     R   0/000000DE   FCNT.     XR  X/00000000
K1$CLKI   XR  X/00000000   K1$SERR   XR  X/00000000   K2$CHAR   XR  X/00000000
K2$CHRI   XR  X/00000000   K2$PINT   XR  X/00000000   PDID      ER    50444F53
SLV$      XR  X/00000000
```

1.3 MBIOS SWITCHES

Execute SY$STRT on boot up	AS=Auto Start
Initial baud rate	BR
Highest memory address	HR undefined
ANSI 3.64 PSC/CSC	ANS=1
Directory flag	FDR=0
Tics/second (system dependent)	TPS=100
Default disk number	SD=0
System flags	SF=0
Clear screen and position cursor code	CPSP Long=$AA00 =$9B3D
Event 112	EV112=TPS/5
Event 113	EV113=1
Event 114	EV114=10
Event 115	EV115=20
Mail array size	MZ=256
RAM disk unit	RU=8
RAM disk size	RZ=255
Number of directory entries	RE=8
RAM disk address	RA=0
RAM disk initialization	IRD=1

(1.3 MBIOS SWITCHES continued)

The switches are described in detail following. Most of them have default values.

AS — AUTO START. This switch determines whether or not the SY$STRT (auto start) file is to be executed on startup. If AS=0, then the SY$STRT file is not executed. If it is non-zero, it is executed. Default=1.

BR — INITIAL BAUD RATE. This is a number from 0 to 7 which represents the initial baud rate for the character I/O ports. The default is 0.

0=19200	4=1200
1=9600	5=600
2=4800	6=300
3=2400	7=110

HR — HIGHEST MEMORY ADDRESS. The high memory address variable determines whether memory is sized (HR undefined) or fixed (HR=top address). Default is undefined.

ANS — ANSI 3.64 PSC/CSC. If this switch is equal to 1, then the BIOS subroutine for clear screen and position cursor for ANSI 3.64 terminal support is included. Default=1.

FDR — DIRECTORY FLAG. The directory flag determines the mode of access for the file manager. When the flag is zero (plus byte), all levels are global. When the flag is set to $80 (minus byte), then files are unique to each directory level. The only exception is level 0 which is global to all. Default is 0, for soft level partitioning.

/FDR=0 global levels
/FDR=$80 unique levels

TPS — TICS/SECOND. The tics/second variable sets the number of clock interrupts that are equivalent to one second. Default is system dependent. You cannot vary this switch without altering B$CPU clock chip initialization.

SD — DEFAULT DISK #. The default disk number determines which disk number is selected when no disk is specified by a filename. Default=0 and may be altered by the B$RSW routine.

Continued on next page...

(3.1 MBIOS SWITCHES continued)

SF	SYSTEM FLAGS. The system flags are used by PDOS to control various output formats. Default=0.
CPSC LONG	CLEAR SCREEN AND POSITION CURSOR CODE. The clear screen codes are used by the XCLS primitive. Default=$AA00. The position cursor codes are used by the XPSC primitive. Default=$093D. Set CPSC=$AA00093D.
EV112	EVENT 112. The event 112 variable is decremented every clock interrupt. Default=TPS/20.
EV113	EVENT 113. The event 113 variable is decremented every second. Default=1.
EV114	EVENT 114. The event 114 variable is decremented every second. Default=10.
EV115	EVENT 115. The event 115 variable is decremented every second. Default=20.
MZ	MAIL ARRAY SIZE. The MAIL array size is in bytes. Default=256 and is best as a multiple of 256.
RU	RAM DISK UNIT. The RAM disk unit is selected by 'RU= '. Default=8.
RZ	RAM DISK SIZE. The size of the RAM disk determines how much memory to allocate. If RZ=0, then no RAM disk is selected. Default=255.
RE	# OF DIRECTORY ENTRIES. The number of directory entries in the RAM disk is selected by 'RE= '. Default=32.
RA	RAM DISK ADDRESS. The ADDRESS variable determines where the RAM disk is located. If RA=0, then the RAM disk is allocated off the top of memory. Otherwise, the parameter indicates the memory address of a RAM disk. Default=0.
IRD	RAM DISK INITIALIZATION. The RAM disk will be initialized by the PDOS BIOS by setting IRD=1 (default).

2. xxBIOSU - UART DSRs

The UART Device Service Routines are supplied in the xxBIOSU:SR module. Up to four different types of UARTs can be used in any one PDOS system. Each UART type is called via a branch table. An entry is provide for get character, put character, baud UART, reset UART, read UART status, high, and low water.

The table is as follows:

```
        U$xDSR  BRA.S   UxDG            ;GET CHARACTER
                BRA.S   UxDP            ;PUT CHARACTER
                BRA.S   UxDB            ;BAUD UART
                BRA.S   UxDR            ;RESET UART
                BRA.S   UxDS            ;READ UART STATUS
                BRA.S   UxHW            ;HIGH WATER
                BRA.S   UxLW            ;LOW WATER
```

An annotated boiler plate follows for a single UART type. Other types follow the same pattern.

```
        TTL     xxBIOSU:SR - 68K UARTS BIOS
*       xxBIOSU:SR      04/12/84
*****************************************************************************
*
*       UU      UU        AA        RRRRRRRR   TTTTTTTT   SSSSSS
*       UU      UU       AAAA       RR     RR     TT     SS
*       UU      UU      AA  AA      RR     RR     TT     SS
*       UU      UU     AAAAAAAA     RRRRRRRR      TT      SSSSSS
*       UU      UU    AA      AA    RR   RR       TT           SS
*       UU      UU   AA        AA   RR    RR      TT           SS
*        UUUUUUU    AA          AA  RR     RR     TT     SSSSSSS
*
*=***************************************************************************
*=      REVISION SCHEDULE MODULE: xxBIOSU
*=
*=
xxBIOSU         IDNT    3.0     M68000 PDOS
*=
*=***************************************************************************
*
        XDEF    U$1DSR
        XDEF    U.1ADR,U.1TYP
*
        OPT     CRE,ALT
        PAGE
        INCLUDE xxPARM:SR
        TTL     xxBIOSU:SR - 68K UARTS BIOS
        SECTION 14
```

(2. xxBIOSU - UART DSRs continued)

```
**************************************************
*       PDOS CHARACTER I/O ROUTINES
**************************************************
*
* EACH UART ENTRY IS DEFINED AS FOLLOWS:
*
*       U$xDSR  BRA.S   UxDG            ;GET CHARACTER
*               BRA.S   UxDP            ;PUT CHARACTER
*               BRA.S   UxDB            ;BAUD UART
*               BRA.S   UxDR            ;RESET UART
*               BRA.S   UxDS            ;READ UART STATUS
*               BRA.S   UxHW            ;HIGH WATER
*               BRA.S   UxLW            ;LOW WATER
*
*       UARTS:   0(A2) = GET A CHARACTER  OUT: A0=BASE, D0=CHAR
*                2(A2) = PUT A CHARACTER   IN: A0=BASE, D0=CHAR, SR=^S^Q
*                4(A2) = BAUD THE PORT     IN: A0=BASE, D0=BAUDRATE
*                6(A2) = RESET THE PORT    IN: A0=BASE
*                8(A2) = READ PORT STATUS  IN: A0=BASE
*               10(A2) = HIGH WATER        IN: A0=BASE, D1=FLAGS
*               12(A2) = LOW WATER         IN: A0=BASE, D1=FLAGS
*
*               F8BT. = FHPI 8DCS
BCSQ    EQU     0       ;\\\\ \\\\_________ 0 = ^S^Q ENABLE
BINC    EQU     1       ; \\\\ \\\_________ 1 = CONTROL CHARACTER DISABLE
BDTR    EQU     2       ;  \\\\ \\_________ 2 = DTR ENABLE
B8CH    EQU     3       ;   \\\\ \_________ 3 = 8 BIT CHARACTER ENABLE
BRIN    EQU     4       ;    \\\\__________ 4 = RECEIVER INTERRUPTS DISABLE
BEVP    EQU     5       ;     \\\__________ 5 = EVEN PARITY ENABLE
BHLW    EQU     6       ;      \\__________ 6 = HIGH/LOW WATER (RESERVED)
BFSQ    EQU     7       ;       \__________ 7 = ^S^Q FLAG BIT (RESERVED)
        PAGE
```

(2. xxBIOSU - UART DSRs continued)

```
**************************************************
* UART ENTRIES ARE DEFINED AS FOLLOWS:
*
*      UxDG - GET CHARACTER
*
*                 OUT: D0.B = CHARACTER
*                      A0.L = UART BASE ADDRESS
*                        SR = EQ....CHARACTER FOUND
*                             NE....NO CHARACTER FOUND
*                             CS....CHARACTER FOUND BUT IGNORE
*
*              NOTE: 1) ALL UARTS OF THE SAME TYPE MUST BE CHECKED
*                       FOR A CHARACTER.
*                    2) PRESERVE & RESTORE ALL REGISTERS USED.
*
*      UxDP - PUT CHARACTER
*
*                  IN: D0.B = CHARACTER
*                      D1.B = PORT FLAG (xxPI 8DBS)
*                      A0.L = UART BASE ADDRESS
*
*              OUT:      SR = .EQ. SENT
*
*              NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*      UxDB - BAUD UART
*
*                  IN: D0.W = BAUD RATE (0-7)
*                      D1.B = PORT FLAG (xxPI 8DBS)
*                      A0.L = UART BASE ADDRESS
*                 OUT:   SR = EQ....UART SUCCESSFULLY BAUDED
*                             NE....UART NOT SUCCESSFULLY BAUDED
*
*              NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*      UxDR - RESET UART
*
*                  IN: A0.L = UART BASE ADDRESS
*                 OUT:   SR = EQ....UART SUCCESSFULLY RESET
*                             NE....UART NOT SUCCESSFULLY RESET
*
*              NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*      UxDS - READ UART STATUS
*
*                  IN: A0.L = UART BASE ADDRESS
*                 OUT: D0.W = UART STATUS
*
*              NOTE: PRESERVE & RESTORE ALL REGISTERS.
```

(2. xxBIOSU - UART DSRs continued)

```
***************************************************
*
U.1ADR EQU      <address>       ;UART BASE ADDRESS
*
U1HW   MOVE.L  D0,-(A7)         ;SAVE D0
       MOVEQ.L #'S'-'@',D0
       BRA.S   WOUT
*
U1LW   MOVE.L  D0,-(A7)
       MOVEQ.L #'Q'-'@',D0
*
WOUT   BTST.L  #BCSQ,D1         ;^S^Q?
         BEQ.S @0004            ;N
*
@0002  BSR.S   U1DP             ;Y, SEND CHARACTER
         BNE.S @0002
*
@0004  MOVE.L  (A7)+,D0         ;Y, RESTORE D0
       RTS
*
U$1DSR BRA.S   U1DG             ;GET A CHARACTER      A0=BASE D0=CHAR
       BRA.S   U1DP             ;PUT A CHARACTER      A0=BASE D0=CHAR
       BRA.S   U1DB             ;BAUD THE PORT        A0=BASE D0=S/BAUDRATE
       BRA.S   U1DR             ;RESET THE PORT       A0=BASE
       BRA.S   U1DS             ;READ PORT STATUS     A0=BASE D0=STATUS
       BRA.S   U1HW             ;HIGH WATER
       BRA.S   U1LW             ;LOW WATER
*
***************************************************
*      RESET UART
*
U1DR   EQU     *                ;RESET UART
```

This routine is called to disable a UART from interrupting.

```
       BRA.S   U1_EQ            ;GOOD RETURN
```

(2. xxBIOSU - UART DSRs continued)

```
**************************************************
*       PUT CHARACTER
*
U1DP    BTST.B  #BDTR,D1
          BNE.S U1DP02           ;NO DTR CHECK
```

If the UART has a DTR input signal, it should be checked here indicating if the device wants the output stream to stop temporarily (such as with a buffer full condition).

```
          BNE.S U1_NE            ;NO DTR
*
U1DP02 BTST.B  <Busy>           ;Y, CAN WE OUTPUT A CHAR?
          BEQ.S U1_NE            ;N, SEND .NE.
```

The character in data register D0.B is output to the UART.

```
        BRA.S   U1_EQ            ;RETURN .EQ.
*
**************************************************
*       GET CHARACTER & RESET INTERRUPT
*
U1DG    LEA.L   U.1ADR,A0        ;GET PORT 1 BASE
        BTST.B  <Character>      ;IS CHARACTER THERE?
          BEQ.S U1_NE            ;N, SEND .NE.
```

All UARTs of the same type should be checked here for a character. If found, the UART base address is returned in address register A0 and the character in data register D0.B. The status register returns the results of the character poll.

```
*
U1_EQ   CMP.B   D0,D0            ;SET .EQ.
        RTS                      ;RETURN
*
U1_NE   CLR.W   -(A7)            ;SET .NE.
        RTR
```

(2. xxBIOSU - UART DSRs continued)

```
**************************************************
*       READ PORT STATUS
*
U1DS    EQU     *                 ;READ PORT STATUS
```

The UART status is return in data register D0.W and 68000 status register.

```
        RTS                       ;RETURN
*
**************************************************
*       BAUD PORT
*
*       IN:     D0=RATE (0-7)
*               D1.L=(OUT EVENT #(80-95)|PORT FLAGS.B
*               A0.L=Base
*       OUT:      SR=.NE. >> bad base
*                 SR=.EQ. >> baud OK
*
U1DB    EQU     *                 ;BAUD UART
```

The UART is bauded according to data register D0.W. Interrupts are enabled for receive only. The eight bit flag should be observed in deciding whether to send and receive 7 or 8 bit characters.

```
        BRA.S   U1_EQ             ;GOOD RETURN (SR=EQ)
*
        END
```

3. xxBIOSW - READ/WRITE DISK DSRs

The Read/Write sector routines are supplied in the xxBIOSW:SR module. Four entries are supplied for read, write, initialize, and check for floppy motor off. An additional entry is for an error message list used by the PDOS monitor module to report disk errors.

The entry points are as follows:

W$XWSE - Write sector
W$XRSE - Read sector
W$XDIT - Initialize disks
W$XDOF - Check for disk off
W$ERM - Error message list

An annotated boiler plate follows for the xxBIOSW module.

```
        TTL     xxBIOSW:SR - 68K R/W SECTOR BIOS
*       xxBIOSW:SR      05/07/84
******************************************************************************
*
* DDDDDDD   IIII   SSSSSS   KK    KK       RRRRRRRR \\  WW               WW
* DD    DD   II   SS        KK   KK        RR     RR \\  WW             WW
* DD     DD  II   SS        KK KK          RR     RR  \\  WW           WW
* DD     DD  II    SSSSSS   KKKKK          RRRRRRRR    \\  WW    WWW    WW
* DD     DD  II         SS  KK  KK         RR   RR      \\  WW  WW WW  WW
* DD    DD   II         SS  KK   KK        RR    RR      \\  WWWW   WWWW
* DDDDDDD   IIII  SSSSSSS   KK    KK       RR     RR      \\  WW     WW
*
*=****************************************************************************
*=      REVISION SCHEDULE MODULE: SBIOSW
*=
*=
xxBIOSW         IDNT    3.0     M68000 PDOS
*=
*=****************************************************************************
*       PDOS R/W SECTOR MODULE
*
        XDEF    W$XWSE,W$XRSE
        XDEF    W$XDIT,W$XDOF
        XDEF    W$ERM
```

(3. xxBIOSW - READ/WRITE DISK DSRs continued)

```
**************************************************
*       INITIALIZE DISKS
*
W$XDIT EQU     *               ;INITIALIZE DISKS
```

The disk controllers are initialized. Any memory tables or communication variables are also set to a known state.

```
        RTS
*
**************************************************
*       DISK OFF ROUTINE
*
W$XDOF EQU     *               ;DISK OFF
```

This routine is called once every second from the PDOS kernel. It is intended for controllers of 5 1/4" floppies where the motor is turned off after a certain length of time with no access.

```
        RTS
*
**************************************************
*       WRITE SECTOR
*
*       IN:     D0.W = DISK #
*               D1.W = LOGICAL SECTOR #
*               (A2) = BUFFER ADDRESS
*       OUT:      SR = EQ...WRITE COMPLETE
*                      NE...D0.L = ERROR
*
W$XWSE EQU     *               ;WRITE SECTOR
```

The write sector routine outputs the logical 256-byte sector pointed to by address register A2 to the disk. Data register D0.W selects the disk number and register D1.W is the logical sector number. The status is returned EQUAL if the operation completed with no error. Otherwise, a NOT EQUAL status is returned with D0.L containing the error number.

```
        MOVEQ.L #0,D0          ;SET STATUS .EQ.
        RTS
```

(3. xxBIOSW - READ/WRITE DISK DSRs continued)

```
**************************************************
*       READ SECTOR
*
*       IN:     D0.W = DISK UNIT #
*               D1.L = LOGICAL SECTOR #
*               (A2) = BUFFER ADDRESS
*       OUT:      SR = EQ...WRITE COMPLETE
*                      NE...D0.L = ERROR
*
W$XRSE EQU      *               ;READ SECTOR
```

The read sector routine reads the logical 256-byte sector from a disk into the memory buffer pointed to by address register A2. Data register D0.W selects the disk number and register D1.W is the logical sector number. The status is returned EQUAL if the operation completed with no error. Otherwise, a NOT EQUAL status is returned with D0.L containing the error number.

```
       MOVEQ.L #0,D0            ;SET STATUS .EQ.
       RTS
*
**********************************************************
*       COMMON ERROR NUMBERS
*
ERR100 MOVEQ.L #100,D0          ;ILLEGAL DISK #
       RTS                      ;RETURN .NE.
*
ERR101 MOVEQ.L #101,D0          ;SECTOR TOO LARGE
       RTS                      ;RETURN .NE.
*
```

(3. xxBIOSW - READ/WRITE DISK DSRs continued)

```
**************************************************
*       ERROR MESSAGE LIST
*
W$ERM   DC.W    100                             ;Error list bias
        DC.B    'Illega',-'l','drive',0         ;100 Common errors
        DC.B    'Secto',-'r','to',-'o','big',0  ;101
        ....

        DC.B    -1                              ;End
```

3.1 PDOS WINCHESTER STANDARD

The PDOS Winchester standard keeps all the information about the Winchester drive on the Winchester drive. This allows you to 1) use a drive with any number of heads and cylinders, 2) divide up the drive into any combination of large and small partitions, and 3) automatically skip all tracks with manufacturing defects. The PDOS Winchester standard information is contained in a block of data that resides in one or two sectors (usually sector 0) of physical track 0 on each Winchester drive in the system. The Drive Data Block (DDB) consists of three parts:

1) the drive parameters,
2) a variable length partition definition table, and
3) a variable length bad track list.

These tables are built and written to the drive by the xxFRMT utility. They are then read into the parameter RAM area by the xxBIOSW disk initialize subroutine, W$XDIT, and subsequently used by the read/write sector code, W$XRSE and W$XWSE, in the xxBIOSW disk module.

The following discussion of the PDOS Winchester standard uses a strict definition of terms. These definitions are found in the glossary (Appendix G) of this manual.

3.1.1 SYSTEM INDEPENDENT DRIVE PARAMETERS

To allow the use of any size Winchester drive in the PDOS system, the drive parameters are read in from the drive itself. These include the number of heads and cylinders. During disk initialization, if a SCSI (SASI) controller is used in the system, either a 'Set Drive Parameters' or an 'Initialize Drive Characteristics' command is sent to the SCSI controller using the number of heads and cylinders specified in the disk's header sector. Thus, any drive in any PDOS system could actually have any number of heads or cylinders, limited only by the controller or hardware.

3.1.2 DISK PARTITIONS ON DRIVE HEADER

Each PDOS Winchester standard drive has all the necessary disk partition information in the header data. There is a three-word entry for each partition of the drive, consisting of a PDOS disk number, a logical base track, and a logical top track number. The PDOS read/write sector routines in xxBIOSW try to match the requested logical disk number to the disk number associated with a disk partition on an installed Winchester drive. The partition's associated base and top tracks are used to bias the requested PDOS sector number to an actual physical or SCSI logical block number. The number of partitions possible on any one drive or system may be limited by: 1) the amount of data read in by W$XDIT; 2) the data written out by xxFRMT; or 3) the amount of room in low parameter RAM. See the source code or the Installation and Systems Management guide for effective limits.

3.1.3 BAD TRACK MAPPING

Following the partition information in the drive's header is an optional bad track list. This table consists of word entries in increasing order of physical track numbers that should not be accessed (skip them). The logical track number is incremented one for each bad track that is numbered lower than or equal to the requested track. The result is a mapped physical track that corresponds to the requested logical track number, where the physical track number is greater than or equal to the logical track number.

3.1.4 DRIVE DATA BLOCKS (DDBs)

Each PDOS system allocates, in its system parameter RAM, a table of six Drive Data Block addresses -- two for floppy drives and four for Winchester drives. The addresses of the Drive Data Blocks are stored by the xxBIOSW disk initialize routine, W$XDIT, when PDOS first starts up. The actual DDBs are usually stored in the system's parameter RAM area by W$XDIT immediately following the six addresses of the P$PARM table.

If more than one type of disk controller is possible in a particular system, then the general W$XDIT routine calls the individual XDIT routines for each controller installed. These routines usually initialize the controller, and then loop through all possible drive select codes, looking for drives (floppy or Winchester) that may be attached.

(3.1.4 DRIVE DATA BLOCKS continued)

As a floppy disk drive is found, its DDB is stored in one of the first two addresses. Each floppy Drive Data Block is built without accessing the drive, using default parameters, since the floppy drives are common to each system, have only one partition, and don't have bad tracks. If there is only one floppy controller in a system, the only difference between the F0 and F1 tables is usually the drive select byte and the disk number, which is set to 0 and 1, respectively.

As Winchesters are found installed (no read error), then W$XDIT determines if the header data is actually PDOS Winchester standard information. The test for this is that the first four bytes of the header information are 'ME4U' and the next word, signifying the number of heads on the drive, is from one through 16. If it is okay, then the data is moved into a DDB in system parameter RAM and the address is saved in the next available P$PARM table location. If the drive is installed but the header data is not PDOS Winchester standard information, then W$XDIT moves down some default drive data into the DDB in P$PARM.

The four Winchester Drive Data Blocks are filled as W$XDIT finds them in the system, altering the controller number and drive select bytes to match where the drive is found. The first Winchester's Drive Data Block is usually read into the system's parameter RAM area by W$XDIT immediately following the two floppy DDBs. It is referred to as drive 'W0', but it may be attached to any controller with any drive select jumper. The Drive Data Block for drive 'W1' would follow the 'W0' bad track table, and so on. You must be sure that the parameter RAM definition file, xx$PARM:SR, and the system memory map allocate enough room for all the drives that may be installed in the system.

3.1.5 PDOS DISK NUMBERING

PDOS disk numbers 0 and 1 are reserved for floppy drives; disk numbers 2 and above are for Winchester partitions. These Winchester partitions, numbered 2-99, are biased by one track worth of sectors (e.g. 32, 33, 34, 38, or 64). To access sectors in the first track, or base track, of the partition, you use the PDOS disk number plus 100. For example, reading from disk 102 accesses the unbiased disk 2 sectors. If there are 32 sectors per track, then disk 2, sector 0 accesses the same sector as disk 102, sector 32. All of the disk accesses for disks 2-99 and 102-199 use the bad track table of the corresponding drive to offset requested tracks.

(3.1.5 PDOS DISK NUMBERING continued)

The PDOS Winchester standard also defines a way to access all the sectors on a drive, ignoring the bad track table remapping feature. This is needed by the "verify" process in the xxFRMT utility -- to check all the sectors on a track to find new bad tracks. PDOS disk numbers 200-209 are mapped to the physical sectors of drive W0, numbers 210-219 are mapped to drive W1, and so on. Disk 200, sectors 0 through 65535 (0 to $FFFF) access Winchester drive W0 physical sectors 0 through 65535. Disk 201, sectors 0 through 65535 access Winchester drive W0 physical sectors 65536 through 131071 ($10000 to $1FFFF). This pattern continues until disk 209 maps to sectors $90000 to $9FFFF. This will accommodate drives up to 168 Mbytes, formatted. If larger disks must be accessed, then you must alter the xxBIOSW:SR code so that the xxFRMT utility can verify the entire drive. This could be done by consolidating drives: 200-219 are drive W0, 220-239 are drive W1.

Currently disk numbers and partitions for each drive are defined by the format utility, xxFRMT. The partitions on each drive get consecutive disk numbers, starting at a specified number, and skipping the standard RAM disk number, 8. Normally the first partition on drive W0 is assigned PDOS disk number 2. The first partition on drive S1 would normally be assigned the next PDOS disk number higher than the last disk number on drive W0, etc.

APPENDIX A

PDOS ERROR DEFINITIONS

Only PDOS system errors (50-99), assembler errors (300-399), and QLINK errors (500-599) are discussed in this appendix. The BIOS errors can be found in the Installation and Systems Management guide for your hardware system. Language errors are discussed in the reference manual for each specific language. Errors are returned through data register D0 on all assembly primitives.

A.1 PDOS ERROR SUMMARY

PDOS ERROR NUMBERS

PDOS ERR 50 = Illegal name
PDOS ERR 51 = Defined
PDOS ERR 52 = Not open
PDOS ERR 53 = Not defined
PDOS ERR 54 = Type err
PDOS ERR 55 = Fragment
PDOS ERR 56 = EOF
PDOS ERR 57 = Dir full
PDOS ERR 58 = Protected
PDOS ERR 59 = Invalid slot
PDOS ERR 60 = Disk full
PDOS ERR 61 = Already open
PDOS ERR 62 = No start
PDOS ERR 63 = Obj err
PDOS ERR 64 = Section err
PDOS ERR 65 = Unloadable
PDOS ERR 66 = Illegal port
PDOS ERR 67 = Parameter err
PDOS ERR 68 = Not PDOS disk
PDOS ERR 69 = No slot
PDOS ERR 70 = Position err
PDOS ERR 71 = Nesting err
PDOS ERR 72 = Too many tasks
PDOS ERR 73 = No memory
PDOS ERR 74 = No task
PDOS ERR 75 = File locked
PDOS ERR 76 = Task locked
PDOS ERR 77 = Not resident
PDOS ERR 78 = Msg buf full
PDOS ERR 79 = Mem err
PDOS ERR 80 = I/O err
PDOS ERR 81 =
PDOS ERR 82 =
PDOS ERR 83 =
PDOS ERR 84 =
PDOS ERR 85 = Aborted task
PDOS ERR 86 = Phantom port
PDOS ERR 87 =
PDOS ERR 88 =
PDOS ERR 89 =
PDOS ERR 90 = Illegal K2 module primitive
PDOS ERR 91 = Illegal K3 module primitive
PDOS ERR 92 = Illegal F module primitive
PDOS ERR 93 = Illegal W module primitive
PDOS ERR 94 = Illegal N module primitive
PDOS ERR 95 = Illegal D module primitive
PDOS ERR 96 = Illegal M module primitive
PDOS ERR 97 = Illegal B module primitive
PDOS ERR 98 =
PDOS ERR 99 =

PDOS ERROR RANGES

1- 49	BASIC error numbers
50- 99	PDOS system error numbers
100-200	BIOS error numbers (disk)
300-399	MASM error numbers
400-499	C error numbers
500-599	QLINK error numbers
600-699	Pascal error numbers

A.2 PDOS ERROR NUMBERS

ERROR 50 ILLEGAL FILE NAME. Valid file names consist of an alpha character followed by up to 7 alpha-numeric characters. An optional extension and disk number may follow. An extension consists of a colon followed by 1 to 3 characters. A disk number consists of a slash and a number ranging from 0 to 127.

```
>DKDKDKDKF
PDOS ERR 50 Illegal name
>
```

ERROR 51 FILE ALREADY DEFINED. Each file name is unique to a disk file directory. There is one directory per disk number.

```
>DF FILE1
>DF FILE1
PDOS ERR 51 Defined
>
```

ERROR 52 FILE NOT OPEN. An attempt to access a file which has not been opened, results in error 52.

```
>EX
FILE 1,1;3,I
*ERROR 52 Not open
```

ERROR 53 FILE NOT DEFINED. If the file name does not exist in the disk directory, an error 53 occurs.

```
>SF FILE2
PDOS ERR 53 Not defined
>
```

ERROR 54 INVALID FILE TYPE. Valid file types are AC, BN, OB, SY, BX, EX, TX, DR, *, and **. All others result in error.

```
>SA FILE1,TR
PDOS ERR 54 Type err
>
```

ERROR 55 FRAGMENTED. Error 55 results from attempting to define a contiguous file on a disk unit which does not have enough room or is fragmented such that there is not a big enough contiguous block of sectors.

```
>DF FILE2,10000
PDOS ERR 55 Fragment
>
```

ERROR 56 END-OF-FILE. Error 56 comes from an attempt to read past the END-OF-FILE index of a file.

```
>EX
*READY
OPEN "#PAUL",F
FILE 1,F;3,I
*ERROR 56 EOF
```

ERROR 57 DIRECTORY FULL. The file directory size is set when the file is initialized. Any attempt to define another file after the directory has been filled, results in error 57.

```
>DF FILE3
PDOS ERR 57 Dir full
>
```

ERROR 58 FILE DELETE PROTECTED. An attempt to delete a file with a delete or write protect flag results in error 58.

```
>SA TEMP,*
>DL TEMP
PDOS ERR 58 Protected
>
```

(A.2 PDOS ERROR NUMBERS continued)

ERROR 59	INVALID SLOT #. A valid file slot number is returned from PDOS on all open commands. A file slot consists of the the disk number in the left byte and the slot index in the right byte.	>EX *READY FILE 1,F;3,I *ERROR 59 Invalid slot
ERROR 60	DISK SPACE FULL. An attempt to extend a file or define a file after the disk space is filled results in error 60.	>CF TEMP,LIST PDOS ERR 60 Disk full >
ERROR 61	FILE ALREADY OPEN. A file can be opened only once in sequential (XSOP) and random (XROP) modes. Read only open (XROO) and shared random open (XNOP) can be executed more than once on the same file.	>EX *READY OPEN "LIST",F OPEN "LIST",F *ERROR 61 Already open
ERROR 62	NO START ADDRESS. An object (OB) file must have a start address. This is generated by an address parameter for the 'END' statement in the assembly source.	>TEMP PDOS ERR 62 No start >
ERROR 63	ILLEGAL OBJECT TAG. Only hex object tag characters are legal.	>SA TEST:SR,OB >TEST:SR PDOS ERR 63 Obj err >
ERROR 64	ILLEGAL SECTION. Only section 0 is executable under PDOS.	>TEMP PDOS ERR 64 Section err
ERROR 65	FILE NOT LOADABLE. Only files typed 'OB', 'SY', 'EX', and 'BX' are loadable by the monitor loader.	>SA ASM,BN >ASM PDOS ERR 65 Unloadable >
ERROR 66	ILLEGAL PORT NUMBER OR BAUD RATE. Only 1 through 15 are legal ports. Valid baud rates are 110, 300, 600, 1200, 2400, 4800, 9600, and 19200.	>BP 2,1250 PDOS ERR 66 Illegal port >BP 20,9600 PDOS ERR 66 Illegal port >
ERROR 67	INVALID PARAMETER. Most monitor commands check parameters for valid ranges and types.	>IM 0 PDOS ERR 67 Parameter err >

(A.2 PDOS ERROR NUMBERS continued)

ERROR 68	NOT A PDOS DISK. An initialized PDOS disk has the constant >A55A at location >0028 of the header sector (sector 0). If the constant is not found on a disk read, error 68 results.	>LS /2 PDOS ERR 68 Not PDOS disk >
ERROR 69	NOT ENOUGH FILE SLOTS. A maximum of 32 files can be open at a time. These correspond to the 32 file slots.	>CF TEMP,TEMP1 PDOS ERR 69 No slot >
ERROR 70	POSITION ERROR. Error 70 results from a position command beyond the end-of-file index.	>EX *READY OPEN "#PAUL",F FILE 1,F;4,0 *ERROR 70 Position err
ERROR 71	NESTING ERROR. Error 71 results for nesting procedure files too deep.	
ERROR 72	TOO MANY TASKS. The task list is defined when the PDOS system is generated.	>@CF LIST,$TTA PDOS ERR 72 Too many tasks >
ERROR 73	NOT ENOUGH MEMORY. An attempt to create a task with more memory than the current task or available memory in the system memory bit maps, results in error 73.	>CT ,40,,1 PDOS ERR 73 No memory >
ERROR 74	NO SUCH TASK. Error 74 occurs when referencing a task not in the task list or task 0.	>KT 5 PDOS ERR 74 No task >
ERROR 75	FILE LOCKED. Once a file has been locked (XLKF), it cannot be accessed until unlocked (XULF).	>CF FDATA,TEMP PDOS ERR 75 File locked >
ERROR 76	TASK LOCKED. Once a task has been locked (XLKT), it cannot be killed until unlocked (XULT).	>KT 5 PDOS ERR 76 Task locked >
ERROR 77	NOT RESIDENT. If PDOS BASIC is not resident in the system, all 'BX' and 'EX' files will not execute. Also, the interpreter cannot be entered with the 'EX' command.	>EX PDOS ERR 77 Not resident

(A.2 PDOS ERROR NUMBERS continued)

ERROR 78 MESSAGE BUFFER FULL. There are 32 message buffers in the PDOS system. Too many messages results in error 78.

```
>SM 4,ANOTHER MESSAGE
PDOS ERR 78 Msg buf full
>
```

ERROR 79 MEMORY ERROR. Error results from a XFUM primitive with invalid arguments.

ERROR 80 I/O DRIVER ERROR. Driver dependent.

ERROR 81 UNIMPLEMENTED PDOS PRIMITIVE. A defined PDOS primitive is not currently implemented.

ERROR 82 ILLEGAL PDOS PRIMITIVE. An invalid A-line primitive has been executed.

ERROR 83 DELAY EVENT STACK FULL. Too many delayed events have been requested.

ERROR 84 CHECKSUM ERROR. Not implemented.

ERROR 85 ABORTED TASK. If a task is aborted by the scheduler, error 85 results.

ERROR 86 PHANTOM PORT. A task has made a call to get character without any possibility of getting a character.

(A.2 PDOS ERROR NUMBERS continued)

ERROR 90 ILLEGAL K2 MODULE PRIMITIVE. Run module error where a kernel #2 primitive has been executed and the module was not generated in the PDOS system.

ERROR 91 ILLEGAL K3 MODULE PRIMITIVE. Run module error where a kernel #3 primitive has been executed and the module was not generated in the PDOS system.

ERROR 92 ILLEGAL F MODULE PRIMITIVE. Run module error where a file manager primitive has been executed and the module was not generated in the PDOS system.

ERROR 93 ILLEGAL W MODULE PRIMITIVE. Run module error where a R/W module primitive has been executed and the module was not generated in the PDOS system.

ERROR 94 ILLEGAL N MODULE PRIMITIVE. Run module error where a floating point module primitive has been executed and the module was not generated in the PDOS system.

ERROR 95 ILLEGAL D MODULE PRIMITIVE. Run module error where a debugger module primitive has been executed and the module was not generated in the PDOS system.

ERROR 96 ILLEGAL M MODULE PRIMITIVE. Run module error where a monitor module primitive has been executed and the module was not generated in the PDOS system.

ERROR 97 ILLEGAL B MODULE PRIMITIVE. Run module error where a BASIC module primitive has been executed and the module was not generated in the PDOS system.

A.3 MASM ERROR NUMBERS

ERROR 301	ILLEGAL SYMBOL.
ERROR 302	MULTIPLY DEFINED SYMBOL.
ERROR 304	UNDEFINED SYMBOL.
ERROR 305	PHASE ERROR.
ERROR 306	ILLEGAL OPCODE.
ERROR 307	ILLEGAL OPCODE EXTENSION.
ERROR 309	MISSING OPERAND.
ERROR 310	ILLEGAL OPERAND MODE.
ERROR 311	UNARY OPERATOR ERROR.
ERROR 312	STACK UNDERFLOW.
ERROR 313	STACK OVERFLOW.
ERROR 314	SYNTAX ERROR.
ERROR 315	ABSOLUTE EXPRESSION REQUIRED.
ERROR 316	ILLEGAL COMPLEX EXPRESSION.
ERROR 319	DISPLACEMENT FIELD OVERFLOW.
ERROR 320	DIVISION BY ZERO.
ERROR 322	BRANCH TO ODD ADDRESS.
ERROR 324	PARAMETER OUT OF RANGE.
ERROR 325	ILLEGAL REGISTER LIST.
ERROR 327	ILLEGAL SECTION SPECIFICATION.
ERROR 328	ILLEGAL OPTION.
ERROR 329	LABEL NOT ALLOWED.
ERROR 330	IF/ENDC OR MACRO/ENDM ERROR.
ERROR 331	FLOATING POINT ERROR.

WARNINGS

300 Modified instruction
303 Multiply defined symbol referenced
308 Was on odd byte boundary
317 Arithmetic overflow
318 Numeric overflow
321 Unmatched quotes or parens
323 Branch could be shorter
326 String truncated

(A.3 MASM ERRORS continued)

Auxiliary errors are additional information for diagnosing an assembler error. They are generally associated with conditional assembly or macros.

Auxiliary errors

ERROR 332 ENDC WITHOUT MATCHING IFxx.

ERROR 333 ENDM WITHOUT MACRO HEADER.

ERROR 334 LEGAL ONLY IN BODY OF MACRO.

ERROR 335 MACRO LABEL NOT FOUND.

ERROR 336 MUST BE SYMBOL.

ERROR 337 LABEL REQUIRED.

ERROR 338 MACRO DEFINITIONS CANNOT BE NESTED.

ERROR 339 INFINITE PARAMETER SUBSTITUTION.

68020 errors

ERROR 340 68020 INSTRUCTION OR ADDRESS MODE.

ERROR 341 ILLEGAL IS/I MEMORY INDIRECTION.

ERROR 342 EXPECTING CLOSING PARENTHESES.

ERROR 343 EXPECTING COMMA.

ERROR 344 ILLEGAL SCALE FACTOR.

ERROR 345 ILLEGAL {OFFSET:WIDTH} FORMAT.

ERROR 346 ILLEGAL REGISTER SPECIFICATION.

A.4 QLINK ERROR DEFINITIONS

ERROR 501	ILLEGAL COMMAND.
ERROR 502	ILLEGAL NUMBER.
ERROR 503	ILLEGAL SECTION SPECIFICATION.
ERROR 504	ILLEGAL SYMBOL.
ERROR 505	TOO MANY COMMAND FILES.
ERROR 506	PDOS CLOSE ERROR.
ERROR 507	PDOS OPEN ERROR.
ERROR 508	PDOS LOAD ERROR.
ERROR 509	'OB' or 'SY' FILE REQUIRED.
ERROR 510	MEMORY SIZE EXCEEDED.
ERROR 511	ILLEGAL OBJECT TAG.
ERROR 512	INVALID ADDRESS RANGE.
ERROR 513	PDOS READ ERR.
ERROR 514	ILLEGAL OPTION.
ERROR 515	ARITHMETIC OVERFLOW.
ERROR 516	DIVISION BY ZERO.
ERROR 517	PDOS WRITE ERROR.
ERROR 518	ILLEGAL SECTION GROUPING.
ERROR 519	NESTING ERROR.
ERROR 520	FIELD OVERFLOW.
ERROR 521	SYMBOL NOT FOUND.
ERROR 522	SYMBOL ALREADY DEFINED.
ERROR 523	UNDEFINED SYMBOL.
ERROR 524	MEMORY OVERFLOW.

APPENDIX B

USER COMMAND SUMMARY

Current PDOS resident monitor commands:

AC - Review procedure	GM - Get memory	RD - RAM disk
AF - Append file	GO - Execute	RN - Rename file
BP - Baud port	GT - Go to label	RS - Reset
CF - Copy file	HE - Help	SA - Set file attributes
CT - Create task	IA - If altered	SF - Show file
DF - Define file	ID - Init date & time	SM - Send task message
DL - Delete file	IF - Conditional	SP - Disk usage
DM - Delete multiple	KM - Kill message	SU - Spool unit
DN - Download file	KT - Kill task	SV - Save to file
DT - Display time	LL - List levels	SY - System disk
EE - Enable echo	LO - Load file	TF - Transfer files
ER - List error	LS - List directory	TM - Transparent mode
EV - Events	LT - List tasks	TP - Task priority
EX - Basic	LV - Directory level	UN - Output unit
FE - For every	MF - Make file	UP - Upload from port
FM - Free memory	PB - Debugger	ZM - Zero memory
FS - File slots	RC - Reset console	

Monitor command formats are as follows:

AC <file>	Review procedure file
AF <file1>,<file2>	Append file
BP {{-}<prt>,<rt>{,<ty>,<bs>}}	Baud port
CF <file1>,<file2>	Copy file
CT <cmd>,<sze>,<prity>,<prt>	Create task
DF <file>{,<size>}	Define file
DL <file>	Delete file
DM <filelist>{,A}	Delete multiple
DN <file>	Download file
DT	Display time
EE <echo flag>	Enable echo

Continued on next page. . .

(B. USER COMMAND SUMMARY continued)

ER <error#>	List error
EV {{-}<event>}	Events
EX	Basic
FE <fl> or (<s>,<e>),<cmd>	For every
FM {{-}<kbytes>}	Free memory
FS	File slots
GM {<kbytes>}	Get memory
GO {<address>}{,<arguments>}	Execute
GT <label>	Go to label
HE {<list>{,<list>...}	Help
IA <file>.<command>	If altered
ID	Init date & time
IF <str1>{=#<str2>}.<cmd>	Conditional
KM <task#>	Kill message
KT {-}<task#>	Kill task
LL <filelist>	List level
LO <file>{,<start addr>}	Load file
LS {<filelist>}{,<file>}	List directory
LT {<mode>}	List tasks
LV {<level>}	Directory level
MF <file>	Make file
PB	Debugger
RC	Reset console
RD {{-}<unt>,<sze>,<adr>}	RAM disk
RN <file1>,<file2>	Rename file
RS {<disk#>}	Reset
SA <file>{,<attribute>}	Set file attributes
SF {-}<file>	Show file
SM {<task#>,<message>}	Send task message
SP {<disk#>}	Disk usage
SU <unit>{,<file> or <port#>}	Spool unit
SV <file>{,<sadr>,<eadr>}	Save to file
SY {<disk#>...}	System disk
TF <filelist>,<disk#>{,<flag>}	Transfer files
TM {{-}<port#>}{,<break>}	Transparent mode
TP {<task#>,}<priority>	Task priority
UN {<unit#>}	Output unit
UP {<port#>}{,<message>}	Upload from port
ZM	Zero memory

(B. USER COMMAND SUMMARY continued)

```
AC
 Command: Review procedure file
  Format: AC <file name>
   Notes: (Y/N/A) Y=Execute line only
                  N=Don't execute
                  A=Execute the rest of the AC file
AF
 Command: Append file1 to the end of file2
  Format: AF <file1>,<file2>
   Notes: <file1> is not altered
          ^C interrupts transfer
BP
 Command: Baud port
  Format: BP
          BP {-}<port #>
          BP {-}<port #>,<baud rate>
          BP {-}<port #>,<baud rate>,<type>,<UART base addr>
   Notes: If <port #> is negative, U2P$ is set
          If no parameters, then list current configured ports

          <port #>       Port (1-15)
                         Port + $100 = ^S^Q protocol
                         Port + $200 = Pass control characters
                         Port + $400 = DTR protocol
                         Port + $800 = 8-bit character I/O

          <baud rate>    0 = 19200 baud
                         1 = 9600 baud
                         2 = 4800 baud
                         3 = 2400 baud
                         4 = 1200 baud
                         5 = 600 baud
                         6 = 300 baud
                         7 = 110 baud

          <type>         Optional (See PDOS UART module)

          <UART addr>    Optional (See PDOS UART module)

 Example: BP 2,9600                    Baud port 2 at 9600 baud
          BP -$402,1200                Baud port 2 at 1200 baud with
                                       DTR handshaking & set U2P$
          BP 4,0,3,$FFFFC4C1           Baud port 4 at 19200 baud &
                                       bind to type 3 UART at $FFFFC4C1
```

(B. USER COMMAND SUMMARY continued)

```
CF
 Command: Copy <file1> into <file2>
  Format: CF <file1>,<file2>
   Notes: <file1> is not altered
          ^C interrupts transfer
CT
 Command: Create task
  Format: CT <task>,<size>,<time*256+priority>,<port>
   Notes: All parameters are optional

          <task>        Task command line             Monitor
          <size>        Size of new task in K bytes   32 K
          <time>        Time slice constant           4
          <priority>    Task priority (1-255)         64
          <port>        Task I/O port                 0

 Example: CT (MASM FILE:SR,FILE),100        Background assembly
          CT ,300,$540,2                    New user on port 2
DF
 Command: Define file in disk directory
  Format: DF <file>{;<level>}{/<disk>}
          DF <file>{;<level>}{/<disk>},<size>
   Notes: Defines contiguous file of <size> sectors
          <size> defaults to 1 sector
          252 bytes/sector
DL
 Command: Delete file from disk directory
  Format: DL <file>
DM
 Command: Delete multiple files from disk directory
  Format: DM <filelist>{,A}
   Notes: Memory is destroyed by this command
          (Y/N/A) Y=Delete file
                  N=Don't delete
                  A=Delete file and all subsequent files

   Files containing the attribute '*' or '**' must have these attributes
   removed by the SA command before they can be deleted.

          <filelist> = {file}{:ext}{;level}{/disk}{/select...}

                         {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
                         {:ext} = 1 to 3 characters (:@=all,*=wild)
                       {;level} = directory level (;@=all)
                        {/disk} = disk number ranging from 0 to 255
                      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                                  Change date (/Fmm-dy-yr,/Tmm-dy-yr)
                                  or          (/Fdy-mon-yr,/Tdy-mon-yr)

 Example: DM M*:@;7/3/F1-1-84/T12-31-84  Delete all 2 character files beginning
                                         with M, any extension, level 7, disk
                                         3, that were altered in 1984
```

(B. USER COMMAND SUMMARY continued)

```
DN
 Command: Download file to U2P port
  Format: DN <file>
   Notes: Data independent, binary transfer
          ^C aborts command
DT
 Command: Display date and time
  Format: DT
EE
 Command: Enable echo
  Format: EE <ECF$ flag>
   Notes: <echo>=0 Enable all console output
                 1 Disable all console output
                 2 Disable LS header console output
ER
 Command: List error message
  Format: ER <error#>
EV
 Command: Set/reset or list events
  Format: EV
          EV {-}<event>
   Notes: No parameters lists current events
          + event sets
          - event clears
EX
 Command: Enter BASIC environment
  Format: EX {<parameters>...}
   Notes: If no BASIC resident, error 77
```

(B. USER COMMAND SUMMARY continued)

FE

Command: For Every processor

Format: FE <filelist>,<command line>
FE (<start>,<end>),<command line>

Notes: Generates IMP$ commands in upper memory and reduces EUM$
Memory is destroyed by this command

<filelist> = {file}{:ext}{;level}{/disk}{/select...}

{file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
{:ext} = 1 to 3 characters (:@=all,*=wild)
{;level} = directory level (;@=all)
{/disk} = disk number ranging from 0 to 255
{/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
PDOS attribute (/*,/**)
Change date (/Fmm-dy-yr,/Tmm-dy-yr)
or (/Fdy-mon-yr,/Tdy-mon-yr)

<command line> substitution parameters:

&F = Full file name
&N = File name
&E = Extension
&L = Level
&D = Disk
\ = Carriage return
[= Start sublist
] = End sublist

Example: FE (4,10) EE 2[LS ;@/&F/F1-1-86]EE 0	List all files on disks 4-10 that have been altered in 1986
FE @:SR;4 MASM &F,&N:OBJ	Assemble all :SR files into :OBJ files of the same name

FM

Command: Free memory from current task

Format: FM
FM {-}<k bytes>

Notes: If +, memory is deallocated
If -, memory is dropped and not recoverable

(B. USER COMMAND SUMMARY continued)

FS
Command: File slots
Format: FS

List File Slots heading explanation:

Slot	File slot #
Name	File name ; directory level / disk
ST	Channel status
SM	Sector in memory
PT	Channel buffer pointer
SI	Current file sector index
EOF	End-of-file sector index number / bytes in last sector
TN	Task number which locked/opened the file
BF	Channel buffer address+
FLGS	Channel status flags (lock/shared/error)

+ A zero buffer address indicates the buffer has been rolled to disk.

Channel status is defined as follows:

x1xx	Sequential	xx80	Altered
x2xx	Random	xx04	Contiguous
x6xx	Shared random	xx02	Delete protected
xAxx	Read only random	xx01	Write protected

1xxx	Driver in channel
4xxx	File altered
8xxx	Sector altered

GM
Command: Get task memory
Format: GM
GM <k bytes>
Notes: If no parameter, then all available memory is recovered

GO
Command: Begin task execution
Format: GO
GO {,<arguments>...}
GO <address>{,<arguments>...}
Notes: If no address, then execute at last entry address (EAD$)

(B. USER COMMAND SUMMARY continued)

GT
Command: Go to label
Format: GT <label>
Notes: Echo flag (ECF$) is disabled during search (& restored)
Search begins at beginning of procedure file
Labels beginning with '*' are recommended

HE
Command: Help
Format: HE {<list>{,<list>...}}
Notes: Help file name: HLPTX

IA
Command: If altered
Format: IA <file>.<command>
Notes: If <file> has the altered bit set ($0080 of status word)
then Clear altered bit
Continue command line processing
else Get next command line

ID
Command: Init date & time
Format: ID
Notes: The current system & BIOS IDs are displayed

IF
Command: Conditional execution in procedure file
Format: IF <string1>.<command>
IF <string1>=<string2>.<command>
IF <string1>#<string2>.<command>

KM
Command: Kill message
Format: KM <task#>
Notes: Removes ALL messages directed to <task #>

KT
Command: Kill task
Format: KT {-}<task#>
Notes: Only current or spawned task can be killed
If -, then task memory is not deallocated

(B. USER COMMAND SUMMARY continued)

LL
```
 Command: List files by directory level
  Format: LL <filelist>
   Notes: Files are sorted according to level
          Memory is destroyed by this command

          <filelist> = {file}{:ext}{;level}{/disk}{/select...}

                         {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
                         {:ext} = 1 to 3 characters (:@=all,*=wild)
                       {;level} = directory level (;@=all)
                        {/disk} = disk number ranging from 0 to 255
                      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                                  PDOS attribute (/*,/**)
                                  Change date (/Fmm-dy-yr,/Tmm-dy-yr)
                                  or          (/Fdy-mon-yr,/Tdy-mon-yr)

 Example: LL /5                         List all files on disk 5
          LL ;@/3/F1-1-84/T12-31-84     List all files on disk 3
                                        that were altered in 1984
```
LO
```
 Command: Load file into memory
  Format: LO <file>
          LO <file>,<start addr>
   Notes: Loads SY or OB files into memory at <start addr>
          <start addr> defaults to end of TCB
          Objects can be loaded anywhere in memory
```
LS
```
 Command: List directory
  Format: LS <filelist>
          LS <filelist>,<file>
   Notes: <file> parameter forces output to PDOS file
          EE 2 disables header and appends disk # to file name
          # of files listed and corresponding disk storage follow list

          <filelist> = {file}{:ext}{;level}{/disk}{/select...}

                         {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
                         {:ext} = 1 to 3 characters (:@=all,*=wild)
                       {;level} = directory level (;@=all)
                        {/disk} = disk number ranging from 0 to 255
                      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                                  PDOS attribute (/*,/**)
                                  Change date (/Fmm-dy-yr,/Tmm-dy-yr)
                                  or          (/Fdy-mon-yr,/Tdy-mon-yr)

 Example: LS ***;@/4/EX/TX/F1-1-86      List all files with a 3 character
                                        name, no extension, on any level,
                                        on disk 4, of type EX or TX, that
                                        have been altered since 1985
```

(B. USER COMMAND SUMMARY continued)

LT
Command: List tasks
Format: LT
LT <mode>

List Task heading explanation:

Task	{*=current}Task #/parent task #
Prt	Task priority (1-255) (+ indicates SVF$ set)
Tm	Task CPU tics (1 tic=10 ms)
Event	Suspended event(s)
Map	Task map constant
Size	Task size (k bytes)
PC	Program Counter
SR	Status Register
TB	Task control Block
EM	End of memory
I	Input port number
U	Output unit mask
1	Unit 1 port number
2	Unit 2 port number
4	Unit 4 port number
8	Unit 8 port number

Mode information can be requested by including a numeric parameter. Available modes are 1-7.

Mode parameters:

Mode 1: Selects TCB parameters starting with CLP$. The TCB parameters are defined as follows:

```
TCB = <--1-> <--2-> <--3-> <--4-> <--5-> <--6-> <--7-> <--8->
      <--9-> <-10-> <-11-> <-12-> <-13-> <-14-> <-15-> <-16->
```

1	CLP$ Command Line Pointer
2	BUM$ Beginning of User Memory
3	EUM$ End of User Memory
4	EAD$ Entry Address
5	IMP$ Assigned Input Message
6	ACI$ Assigned Console Inputs
7	LEN$/SFI$ Error Register/Spooling Unit File ID
8	FLG$/SLV$/FEC$/0 Task Bit Flags/Directory Level/ File Expansion Count

Continued on next page. . .

(B. USER COMMAND SUMMARY continued)

```
Mode 1: Selects TCB parameters starting with CLP$.  The TCB parameters
        are defined as follows (continued):

     TCB = <--1-> <--2-> <--3-> <--4-> <--5-> <--6-> <--7-> <--8->
           <--9-> <-10-> <-11-> <-12-> <-13-> <-14-> <-15-> <-16->

     9       CSC$/PSC$ Clear Screen/Position Cursor
     10      SDS$/SDK$ Alternate Disks/System Disk
     11      EXT$ XEXT$ Address
     12      ERR$ XERR$ Address
     13      CMD$/TID$/ECF$/CNT$ Command Line Delimiter/Task ID/
                                 Echo Flag/ Column Counter
     14      MMF$/PRT$/SPU$/UNT$ Memory Modified Flag/Input Port #/
                                 Spooling Unit Mask/Output Unit Mask
     15      U1P$/U2P$/U4P$/U8P$ Unit 1/Unit 2/Unit 4/Unit 8 Ports
     16      0/TWO$ Monitor Temps

Mode 2: Lists current executing monitor command (MPB$).
Mode 3: List both modes 1 and 2.
Mode 4: Outputs current contents of floating point register (FPA$)
Mode 5: Lists modes 1 and 4.
Mode 6: Lists modes 2 and 4.
Mode 7: Lists modes 1, 2, and 4 (all modes).
```

```
Example:
 x>LT 1
 Task  Prt Tm  Event   Map  Size   PC       SR   TB       EM       I U 1 2 4 8
  0/0   64  1  97/-128  0   548   0000EB44 0000 0000D800 00096800 1 1 1 2 0 0
   TCB=0000D903 0000F59C 00096800 0000DD00 00000000 00000000 FFFF0000 00010000
       AA009B3D FFFF0A05 00000000 00000000 00000000 03010001 01020000 00000800
 x>LT
 Task  Prt Tm  Event   Map  Size   PC       SR   TB       EM       I U 1 2 4 8
 *0/0   64  2           0   384   00001D08 2004 0000B000 0006B000 1 1 1 0 0 0
  1/0   64  2  99       0   20    00001B42 2000 0006B000 00070000 3 1 3 0 0 0
```

```
LV
 Command: Directory level
  Format: LV
          LV <level>
   Notes: LV without parameter lists current user level
          Level 255 selects all levels
MF
 Command: Make file
  Format: MF <file>
   Notes: A [CR] writes line to file
          Only current line can be edited
          An [ESC] terminates command and closes file
```

(B. USER COMMAND SUMMARY continued)

PB
 Command: Enter PDOS debugger
  Format: PB {<parameters>...}
   Notes: Debugger executes in supervisor mode

A0-7	A-reg	#	Mem IAC
B{#,a}	Lst/def break	#,#	Mem dump
D0-7	D-reg	#,#+	Disassemble
F	68881 regs	#,#,#{WL}	Find B/W/L
{#}G	Go & break	#(0-7	d(Ax)
M	Last dump	#{+-}#	Hex +/-
N#	0=W,1=B,+2=w/o read		
O	Offset	^D	Disassemble
P	PC	-	Open previous
Q	Exit	LF	Open next
R	Reg dump	+#	# + offset
S	Status		
T	Trace	Trace options:	
U	Unit	--------------	
V	Control IAC	F/R/M	Dump
W{s,e}	Window	G	Go
X	Set breaks & exit	T	Running
Z	Reset		

RC
 Command: Reset console
  Format: RC
   Notes: Only the current procedure file is terminated

RD
 Command: RAM disk
  Format: RD
          RD {-}<unit>,<size>,<address>
   Notes: No parameter lists current RAM disk configuration
          -<unit> will automatically initialize with 32 file directory size
          Each 1 K of memory equals 4 RAM disk sectors

```
 Example: x>RD                           List current RAM disk parameters
          Disk=8
          Size=255                       RAM disk size = 255 sectors
          Addr=000ED800
          x>FM -578                      Free (2560-255)/4 = 576.25 sectors
          Addr=0005D000
          x>RD -8,2560,$5D000            Create and init floppy image RAM disk
          x>SP 8
          Files=0/32
          Free=2554,2554
          Used=0/0
          x>
```

(B. USER COMMAND SUMMARY continued)

```
RN
 Command: Rename file
  Format: RN <file1>,<file2>
          RN <file>,<level>
   Notes: A number for the second parameter is a new directory level
RS
 Command: Reset
  Format: RS
          RS <disk #>
SA
 Command: Set file attributes
  Format: SA <file>
          SA <file>,<attribute>
   Notes: Valid file attributes are as follows:

                AC = Procedure file
                OB = 68000 object
                SY = System file
                TX = ASCII text
                BN = Binary file
                EX = BASIC program
                BX = BASIC binary program
                DR = System I/O driver
                 * = Delete protect
                ** = Write protect
SF
 Command: Show file
  Format: SF <file>
          SF -<file>
   Notes: File listing flow controlled with space bar
          File listing automatically pauses after 23 lines
          Lines are clipped to 78 characters
          A minus sign before line supresses clipping and auto pause
SM
 Command: Send task message
  Format: SM
          SM <task#>,<message>
   Notes: No parameter lists current queued messages
SP
 Command: Disk usage
  Format: SP
          SP <disk>
   Notes: The disk usage is defined as follows:

                Files=<files>/<directory size>
                Free=<free sectors>,<largest contiguous block>
                Used=<sectors used>/<sectors allocated>
```

(B. USER COMMAND SUMMARY continued)

```
SU
 Command: Spool unit
  Format: SU <unit>
          SU <unit>,<file>
          SU <unit>,<port#>
   Notes: Spool file is closed and SPU$ reset with 'SU 0'
          <port #> is loaded into corresponding output variables of <unit>

 Example:       x>LT
                Task   Prt Tm  ...   EM      I U 1 2 4 8
                *0/0   64  1   ...000ED800   1 1 1 0 0 0
                x>SU 2,2
                x>UN 3
                x>LT
                Task   Prt Tm  ...   EM      I U 1 2 4 8
                *0/0   64  1   ...000ED800   1 3 1 2 0 0
                x>UN 1
                x>SU 6,4
                x>LT
                Task   Prt Tm  ...   EM      I U 1 2 4 8
                *0/0   64  1   ...000ED800   1 1 1 4 4 0
                x>
SV
 Command: Save to file
  Format: SV <file>
          SV <file>,<sadr>,<eadr>
   Notes: A binary memory image is written to file
          Default memory bounds are TBE$ and BUM$
SY
 Command: System disk
  Format: SY
          SY <disk>{,<disk>}{,<disk>}{,<disk>}
TF
 Command: Transfer files
  Format: TF <filelist>,<disk #>
          TF <filelist>,<disk #>,<flag>
   Notes: Memory is destroyed by this command

          <filelist> = {file}{:ext}{;level}{/disk}{/select...}

                         {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
                         {:ext} = 1 to 3 characters (:@=all,*=wild)
                       {;level} = directory level (;@=all)
                        {/disk} = disk number ranging from 0 to 255
                      {/select} = PDOS type (/AC,/BN,/BX,/EX,/OB,/SY,/TX,/DR)
                                  PDOS attribute (/*,/**)
                                  Change date (/Fmm-dy-yr,/Tmm-dy-yr)
                                  or          (/Fdy-mon-yr,/Tdy-mon-yr)
```

Continued on next page. . .

(B. USER COMMAND SUMMARY continued)

```
TF (continued)

           <flag> = A,D,U

                  A = Transfer all the files in the filelist.
                  D = Transfer those files in the filelist that are
                      defined on the destination disk.
                  U = Transfer those files in the filelist that are
                      undefined on the destination disk.
TM
 Command: Transparent mode
  Format: TM
          TM {-}<port>
          TM {-}<port>,<break>
   Notes: No parameters defaults to U2P$ and [ESC] for break
          Memory is destroyed by this command with the - option
          If negative port, then on break, prompt for file

 Example:       x>TM 4,2                 Transparent on port 2 with ^B break
                x>TM -4                  Capture data from port 2
TP
 Command: Task priority
  Format: TP <time*256+priority>
          TP <task#>,<time*256+priority>
   Notes: One parameter defaults to current task
          If time is omitted (ie. time=0) then task time is unaltered

 Example:       x>TP 2,100               Task 2 priority set to 100
                x>TP 0,$440              Task 0 time=4, priority=64
UN
 Command: Output unit
  Format: UN
          UN <unit>
   Notes: Each bit of UNT$ selects an output path
UP
 Command: Upload from port
  Format: UP
          UP <port #>
          UP <port #>,<message>
   Notes: The <message> is first sent out port if included
          Data is loaded into user memory from port
          Default port is U2P$
          After each 256 characters, a period is output
          An [ESC] from console or input or timeout terminates transfer
          Memory is destroyed by this command
ZM
 Command: Zero memory
  Format: ZM
   Notes: EAD$ & BUM$ are reset to TBE$
          MMF$ is cleared
```

APPENDIX C

PRIMITIVE COMMAND SUMMARY

```
                            ***************************************************
                            *       PDOS CALL DEFINITIONS
                            ***************************************************
                            *
0/00000000:A000             S       XSWP              ;SWAP TO NEXT PROCESS
0/00000002:A002                     XSMP              ;SEND MESSAGE POINTER
0/00000004:A004                     XGMP              ;GET MESSAGE POINTER
0/00000006:A006                     X881              ;68881 ENABLE
0/00000008:A008                     DC.W     $A008    ;XUSP$ = RETURN TO USER MODE
0/0000000A:A00A                     DC.W     $A00A    ;XPAD$ = PACK ASCII DATE
0/0000000C:A00C                     XERR              ;MONITOR ERROR CALL
0/0000000E:A00E                     XEXT              ;EXIT TO MONITOR
0/00000010:A010                     XGML              ;GET MEMORY LIMITS
0/00000012:A012                     XRTS              ;READ TASK STATUS
0/00000014:A014                     XLKT              ;LOCK TASK
0/00000016:A016                     XULT              ;UNLOCK TASK
0/00000018:A018                     XSEF              ;SET EVENT FLAG
0/0000001A:A01A                     XTEF              ;TEST EVENT FLAG
0/0000001C:A01C                     XSUI              ;SUSPEND UNTIL INTERRUPT
0/0000001E:A01E                     XGTM              ;GET TASK MESSAGE
0/00000020:A020                     XSTM              ;SEND TASK MESSAGE
0/00000022:A022                     XGTP              ;GET TASK PARAMETERS
0/00000024:A024                     XDTV              ;DEFINE TRAP VECTORS
0/00000026:A026                     XCTB              ;CREATE TASK
0/00000028:A028                     XKTM              ;KILL TASK MESSAGE
0/0000002A:A02A                     XRDM              ;DUMP REGISTERS
0/0000002C:A02C                     XSUP              ;MOVE TO SUPERVISOR MODE
0/0000002E:A02E                     XLSR              ;LOAD STATUS REGISTER
0/00000030:A030                     XEXC              ;EXECUTE PDOS CALL D7.W
0/00000032:A032                     XDEV              ;DELAY EVENT
0/00000034:A034                     XRTP              ;READ TIME PARAMETERS
0/00000036:A036                     XUAD              ;UNPACK ASCII DATE
0/00000038:A038                     XBUG              ;CALL DEBUGGER
0/0000003A:A03A                     XLER              ;LOAD ERROR REGISTER
0/0000003C:A03C                     XSTP              ;SET/READ TASK PRIORITY
0/0000003E:A03E                     XGUM              ;GET USER MEMORY
0/00000040:A040                     XFUM              ;FREE USER MEMORY
0/00000042:A042                     XRSR              ;READ STATUS REGISTER
0/00000044:A044                     XRTE              ;INTERRUPT RETURN FROM EXCEPTION
0/00000046:A046                     XSEV              ;SET/RESET EVENT W/O SWAP
0/00000048:A048                     XGCB              ;CONDITIONAL GET CHARACTER
0/0000004A:A04A                     XDMP              ;*DUMP MEMORY F/STACK
0/0000004C:A04C                     XEXZ              ;EXIT TO MONITOR W/COMMAND
0/0000004E:A04E                     XPCB              ;PUSH COMMAND TO BUFFER
```

(APPENDIX C PRIMITIVE COMMAND SUMMARY continued)

```
                            **************************************************
                            *       SUPPORT CALLS
                            *
0/00000050:A050                     XCBD            ;CONVERT BINARY TO DECIMAL
0/00000052:A052                     XCBH            ;CONVERT BINARY TO HEX
0/00000054:A054FFAA                 XCBM    S       ;CONVERT BINARY TO DECIMAL WITH MESSAGE
0/00000058:A056                     XCDB            ;CONVERT DECIMAL TO BINARY
0/0000005A:A058                     XFTD            ;FIX TIME & DATE INTO R0,R1
0/0000005C:A05A                     XGNP            ;GET NEXT PARAMETER
0/0000005E:A05C                     XRDT            ;READ DATE
0/00000060:A05E                     XRTM            ;READ TIME
0/00000062:A060                     XUDT            ;UNPACK DATE
0/00000064:A062                     XUTM            ;UNPACK TIME
0/00000066:A064                     XWDT            ;WRITE DATE
0/00000068:A066                     XWTM            ;WRITE TIME
0/0000006A:A068                     XCHX            ;CONVERT HEX TO BUFFER
0/Q000006C:A06A                     XCBX            ;CONVERT DECIMAL TO BUFFER
0/0000006E:A06CFF90                 XAIM    S       ;ADD INDEXED MESSAGE
0/00000072:A06E                     XPEL            ;PUT ENCODED LINE
                            *
                            **************************************************
                            *       CONSOLE I/O CALLS
                            *
0/00000074:A070                     XBCP            ;BAUD CONSOLE PORT
0/00000076:A072                     XCBC            ;CHECK FOR BREAK CHARACTER
0/00000078:A074                     XCBP            ;CHECK FOR BREAK OR PAUSE
0/0000007A:A076                     XCLS            ;CLEAR SCREEN
0/0000007C:A078                     XGCC            ;GET CONSOLE CHARACTER CONDITIONAL
0/0000007E:A07A                     XGCR            ;GET CONSOLE CHARACTER
0/00000080:A07C                     XGLB            ;GET LINE IN BUFFER
0/00000082:A07E                     XGLM            ;GET LINE IN MONITOR BUFFER
0/00000084:A080                     XGLU            ;GET LINE IN USER BUFFER
0/00000086:A082                     XGLX            ;GET LINE IN BUFFER W/CONTROL CODES
0/00000088:A084                     XPBC            ;PUT USER BUFFER TO CONSOLE (*R9)
0/0000008A:A086                     XPCC            ;PUT CHARACTER TO CONSOLE
0/0000008C:A088                     XPCL            ;PUT CRLF TO CONSOLE
0/0000008E:A08A                     XPLC            ;PUT LINE TO CONSOLE
0/00000090:A08CFF6E                 XPMC    S       ;PUT MESSAGE TO CONSOLE
0/00000094:A08E                     XPSC            ;POSITION CURSOR
0/00000096:A0900000                 XTAB    0       ;TAB
0/0000009A:A092                     XRCP            ;READ CURSOR POSITION
0/0000009C:A094                     XRPS            ;READ PORT STATUS
0/0000009E:A096                     XPDC            ;PUT DATA TO CONSOLE
0/000000A0:A098                     XPSP            ;PUT SPACE TO CONSOLE
0/000000A2:A09A                     XSPF            ;SET PORT FLAG
0/000000A4:A09CFF5A                 XPEM    S       ;PUT ENCODED MESSAGE TO CONSOLE
0/000000A8:A09E                     XGCP            ;GET CHARACTER FROM PORT
```

(APPENDIX C PRIMITIVE COMMAND SUMMARY continued)

```
                        ***************************************************
                        *       FILE SUPPORT I/O CALLS
                        *
0/000000AA:A0A0                 XFFN            ;FIX FILE NAME
0/000000AC:A0A2                 XLFN            ;LOOK FOR NAME IN FILE SLOTS
0/000000AE:A0A4                 XLST            ;LIST FILE DIRECTORY
0/000000B0:A0A6                 XRDE            ;READ DIRECTORY ENTRY
0/000000B2:A0A8                 XRDN            ;READ DIRECTORY NAME
0/000000B4:A0AA                 XAPF            ;APPEND FILE
0/000000B6:A0AC                 XCHF            ;CHAIN FILE
0/000000B8:A0AE                 XCPY            ;COPY FILE
0/000000BA:A0B0                 XLDF            ;LOAD FILE
0/000000BC:A0B2                 XRCN            ;RESET CONSOLE FILE
0/000000BE:A0B4                 XRST            ;RESET FILES
0/000000C0:A0B6                 XSZF            ;SIZE DISK
0/000000C2:A0B8                 XBFL            ;BUILD FILE LIST
                        *
0/000000C4:A0BA                 XPCR            ;PUT CHARACTER RAW
0/000000C6:A0BC                 DC.W    $A0BC   ;XPCP$ = PUT CHARACTER TO PORT
0/000000C8:A0BE                 DC.W    $A0BE   ;XBER$ = BASIC ERROR CALL
                        *
                        ***************************************************
                        *       DISK SUPPORT I/O CALLS
                        *
0/000000CA:A0C0                 XISE            ;INIT SECTOR
0/000000CC:A0C2                 XRSE            ;READ SECTOR
0/000000CE:A0C4                 XRSZ            ;READ SECTOR ZERO
0/000000D0:A0C6                 XWSE            ;WRITE SECTOR
```

(APPENDIX C PRIMITIVE COMMAND SUMMARY continued)

```
                         **************************************************
                         *       FILE MANAGER CALLS
                         *
0/000000D2:A0C8                  DC.W    $A0C8    ;
0/000000D4:A0CA                  DC.W    $A0CA    ;
0/000000D6:A0CC                  DC.W    $A0CC    ;
0/000000D8:A0CE                  XFAC             ;FILE ALTERED CHECK
0/000000DA:A0D0                  XCFA             ;CLOSE FILE WITH NEW ATTRIBUTES
0/000000DC:A0D2                  XCLF             ;CLOSE FILE
0/000000DE:A0D4                  XDFL             ;DEFINE FILE
0/000000E0:A0D6                  XDLF             ;DELETE FILE
0/000000E2:A0D8                  XLKF             ;LOCK FILE
0/000000E4:A0DA                  XNOP             ;OPEN NON-EXCLUSIVE RANDOM
0/000000E6:A0DC                  XPSF             ;POSITION FILE
0/000000E8:A0DE                  XRBF             ;READ BLOCK
0/000000EA:A0E0                  XRFA             ;READ FILE ATTRIBUTES
0/000000EC:A0E2                  XRLF             ;READ LINE
0/000000EE:A0E4                  XRNF             ;RENAME FILE
0/000000F0:A0E6                  XROO             ;OPEN READ ONLY RANDOM
0/000000F2:A0E8                  XROP             ;OPEN RANDOM FILE
0/000000F4:A0EA                  XRWF             ;REWIND FILE
0/000000F6:A0EC                  XSOP             ;OPEN SEQUENTIAL FILE
0/000000F8:A0EE                  XULF             ;UNLOCK FILE
0/000000FA:A0F0                  XWBF             ;WRITE BLOCK
0/000000FC:A0F2                  XWFA             ;WRITE FILE ATTRIBUTES
0/000000FE:A0F4                  XWLF             ;WRITE LINE
0/00000100:A0F6                  XZFL             ;ZERO FILE
0/00000102:A0F8                  XFBF             ;FLUSH BUFFERS
0/00000104:A0FA                  XKTB             ;KILL TASK
0/00000106:A0FC                  XWFP             ;WRITE FILE PARAMETERS
0/00000108:A0FE                  XRFP             ;READ FILE POSITION
                         *
                         **************************************************
                         *       RESERVED SYSTEM CALLS
                         *
0/0000010A:A100                  DC.W    $A100    ;XSER$ = SR=NE, D0=ERROR RETURN
0/0000010C:A102                  DC.W    $A102    ;XSYS$ = GET SYRAM POINTER
0/0000010E:A104                  DC.W    $A104    ;XCLH$ = SYSTEM CONVERT LONG TO HEX
0/00000110:A106                  DC.W    $A106    ;XCWH$ = SYSTEM CONVERT WORD TO HEX
0/00000112:A108                  DC.W    $A108    ;XCLD$ = SYSTEM CONVERT LONG TO DECIMAL
0/00000114:A10A                  DC.W    $A10A    ;XSSP$ = GET SUPERVISOR STACK POINTER
0/00000116:A10C                  DC.W    $A10C    ;XL2E$ = LEVEL 2 SR=NE, D0=ERROR RETURN
                                 DC.W    $A10E    ;XSPT$ = SET PARENT TASK
```

APPENDIX D

PDOS DISK LAYOUT

The following disk sector listings define the PDOS disk formats including the header sector, directory entries, and data storage.

```
x>MDDUMP
68K PDOS Disk Dump/Alter Utility 05/02/84
  Disk # = 0
  To alter sector, enter "A"; to exit, enter "Z"
  Start Sector = 0
  End Sector = 2

 Sector/Disk=$0000 (0)/0
 000-00F   53 41 47 45 20 50 44 4F 53 20 32 2E 36 64 00 00 SAGE PDOS 3.2..      Disk name
 010-01F   09 40 00 6D 88 00 08 00 00 80 09 40 A5 5A FF FF .@.m.......@%Z..       0940 = Boot sector
 020-02F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................       006D = # of files
 030-03F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................         88 = # of boot sectors
 040-04F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................     000800 = Boot address
 050-05F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................       0940 = # of PDOS sectors
 060-06F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................       A55A = PDOS ID
 070-07F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................       FFFF = Sides/Density
 080-08F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 090-09F   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0A0-0AF   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0B0-0BF   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0C0-0CF   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
 0D0-0DF   FF F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .p..............     1 = Allocated
 0E0-0EF   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................     0 = Free
 0F0-0FF   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

.............

 Sector/Disk=$0002 (2)/0
 000-00F   41 4D 41 5A 49 4E 47 00 00 00 00 05 08 00 00 12 AMAZING.........     41....00 = File name
 010-01F   00 00 00 12 00 12 00 9A 10 1F A8 A2 10 1F A8 A2 ..........("..("     00 00 00 = File extension
 020-02F   41 53 4D 00 00 00 00 00 00 00 00 80 00 00 25 ASM............%             05 = Directory level
 030-03F   00 00 00 00 00 00 00 2E 10 1F A8 A2 10 1F A8 A2 ..........("..("           0800 = Type
 040-04F   42 30 31 00 00 00 00 00 00 00 00 0A 20 00 00 26 B01......... ..&           0012 = Start sector
 050-05F   00 00 00 01 00 01 00 58 10 1F A8 A2 10 1F A8 A2 .......X..("..("           0000 = Free
 060-06F   42 30 31 00 00 00 00 00 53 52 00 0A 02 00 00 28 B01.....SR.....(           0012 = Sectors allocated
 070-07F   00 00 00 04 00 04 00 55 10 1F A8 A2 10 1F A8 A2 .......U..("..("           0012 = EOF sector index
 080-08F   42 30 32 00 00 00 00 00 00 00 00 0A 20 00 00 2D B02......... ..-           009A = # of bytes in last sec
 090-09F   00 00 00 01 00 01 00 5B 10 1F A8 A2 10 1F A8 A2 .......[..("..("     101FA8A2 = Date created
 0A0-0AF   42 30 32 00 00 00 00 00 53 52 00 0A 02 00 00 2F B02.....SR...../     101FA8A2 = Date last updated
 0B0-0BF   00 00 00 04 00 04 00 3D 10 1F A8 A2 10 1F A8 A2 .......=..("..("
 0C0-0CF   42 30 33 00 00 00 00 00 00 00 00 0A 20 00 00 34 B03......... ..4
 0D0-0DF   00 00 00 01 00 01 00 5B 10 1F A8 A2 10 1F A8 A2 .......[..("..("
 0E0-0EF   42 30 33 00 00 00 00 00 53 52 00 0A 02 00 00 36 B03.....SR.....6
 0F0-0FF   00 00 00 04 00 04 00 3F 10 1F A8 A2 10 1F A8 A2 .......?..("..("
```

(APPENDIX D PDOS DISK LAYOUT continued)

```
To alter sector, enter "A"; to exit, enter "Z"
Start Sector = $12
End Sector = $13

Sector/Disk=$0012 (18)/0
000-00F   00 13 00 00 FF FF FF FF 00 00 0D 0E 00 00 04 DC ...............\   0013 = Forward link
010-01F   00 00 00 54 00 00 00 68 23 14 41 4D 41 5A 49 4E ...T...h#.AMAZIN   0000 = Backward link (null)
020-02F   47 20 50 52 4F 47 52 41 4D 00 00 00 1C 14 53 45 G PROGRAM.....SE
030-03F   45 44 3D 00 0B 63 07 1A 63 5C 00 2E 07 08 5C 0D ED=..c..c\....\.
040-04F   17 4E 06 63 00 00 08 63 06 5C 0D 17 4E 00 1C 14 .N.c...c.\..N...
050-05F   57 48 41 54 20 41 52 45 20 59 4F 55 52 20 57 49 WHAT ARE YOUR WI
060-06F   44 54 48 20 41 4E 44 20 4C 45 4E 47 54 48 00 0A DTH AND LENGTH..
070-07F   64 0A 65 00 23 14 50 4C 45 41 53 45 20 57 41 49 d.e.#.PLEASE WAI
080-08F   54 2E 2E 2E 2E 00 0B 00 10 64 5C 01 30 65 5C 01 T........d\.0e\.
090-09F   30 18 66 0A 64 5C 01 30 65 5C 01 30 18 67 0A 64 0.f.d\.0e\.0.g.d
0A0-0AF   65 32 17 68 00 00 08 69 06 5C 00 07 08 6A 06 5C e2.h...i.\...j.\
0B0-0BF   00 07 08 6B 06 60 64 32 5C 01 30 17 40 00 08 6C ...k.`d2\.0.@..l
0C0-0CF   06 5C 01 07 08 6B 5C 01 18 67 06 6C 07 08 6C 06 .\...k\..g.l..l.
0D0-0DF   6C 5C 01 30 07 08 6D 06 6B 07 08 6E 06 5C 01 00 l\.0..m.k..n.\..
0E0-0EF   06 6F 06 5C 01 01 64 07 06 70 06 5C 01 01 65 00 .o.\..d..p.\..e.
0F0-0FF   08 6F 70 18 66 06 5C 01 00 00 1F 70 07 1F 6F 00 .op.f.\....p..o.

Sector/Disk=$0013 (19)/0
000-00F   00 14 00 12 01 5D 00 00 01 04 00 00 1D 71 00 00 .....]........q..  0014 = Forward link
010-01F   1A 6D 64 2E 07 08 6D 06 6D 5C 01 30 07 01 5D 00 .md...m.m\.0..].   0012 = Backward link (null)
020-02F   00 00 FA 00 08 6D 06 5C 01 07 08 6E 06 6E 5C 01 ..z..m.\...n.n\.
030-03F   30 07 1A 6E 65 2C 07 08 6E 06 5C 01 00 00 1A 6D 0..ne,..n.\....m
040-04F   6E 18 67 5C 00 29 07 01 5D 00 00 00 C8 00 1A 6D n.g\.)..]...H..m
050-05F   5C 01 31 5C 00 29 07 01 5D 00 00 02 12 00 1A 6D \.1\.)..]......m
060-06F   5C 01 31 6E 18 67 5C 00 2E 07 01 5D 00 00 02 12 \.1n.g\....]....
070-07F   00 00 1A 6E 5C 01 31 5C 00 29 07 01 5D 00 00 01 ...n\.1\.)..]...
080-08F   86 00 1A 6D 6E 5C 01 31 18 67 5C 00 2E 07 01 5D ...mn\.1.g\....]
090-09F   00 00 01 86 00 00 1A 6D 64 29 07 01 5D 00 00 01 .......md)..]...
0A0-0AF   4A 00 1A 6D 5C 01 30 6E 18 67 5C 00 2E 07 01 5D J..m\.0n.g\....]
0B0-0BF   00 00 01 4A 00 00 21 60 5C 03 32 5C 01 30 07 01 ...J..!`\.2\.0..
0C0-0CF   5D 00 00 03 16 0A 5D 00 00 03 34 0A 5D 00 00 03 ].....]...4.]...
0D0-0DF   5C 00 1A 6E 65 2E 07 01 5D 00 00 01 54 00 1A 6A \..ne...]...T..j
0E0-0EF   5C 01 29 07 01 5D 00 00 01 72 00 00 08 69 06 5C \.)..]...r...i.\
0F0-0FF   01 07 01 5D 00 00 01 68 00 00 1A 6D 6E 5C 01 30 ...]...h...mn\.0
To alter sector, enter "A"; to exit, enter "Z"
Start Sector =
```

APPENDIX E

PDOS I/O DRIVERS

PDOS I/O drivers are an extension of the PDOS file system. If a file's attribute is 'DR', then the PDOS file manager expects the file to be an I/O driver program instead of data.

E.1 DRIVER ENTRY POINTS

PDOS I/O drivers are an extension of the PDOS file system. An I/O driver is designated by the 'DR' file type. I/O driver files contain position independent (self-relocating) code rather than data.

Extension of PDOS file system

'DR' file type

Maximum length = 240+12 bytes

When an I/O driver is opened, closed, read from, written to, or positioned, the PDOS file manager branches into the channel buffer at specific entry points. This requires that the first twelve bytes of the file be reserved for branch instructions and that the driver code and variables be no more than 240 bytes in length.

The following driver entry points must be at the beginning of each driver module:

Driver entry points

```
        SECTION 0
        DC.W    $A55A           ;DRIVER ID
DROP    BRA.S   OPEN            ; 2 OPEN
DRCL    BRA.S   CLOS            ; 4 CLOSE
DRRD    BRA.S   READ            ; 6 READ
DRWR    BRA.S   WRIT            ; 8 WRITE
DRPS    BRS.S   POST            ;10 POSITION
```

The driver must be written in position independent or self-relocating 68000 assembly code. This simply means that while the code is relocatable, there can be no relocatable tags within the object file.

Position independent code

A common way to make the code self-relocating is to generate a base address and then reference each constant within the program as a displacement beyond the base address. PDOS passes the base address of the driver buffer in address register A2. This can be conveniently used as the base register for variables defined as the label minus the start address plus four. The former makes the label absolute (relocatable-relocatable=absolute) and the latter skips the file links.

```
        SECTION 0
DTTX    DC.W    $A55A           ;BEGINNING OF DRIVER
        ....

        ADDQ.W  #1,CNT(A2)      ;INCREMENT COUNT
        LEA.L   BUF(A2),A0      ;POINT TO BUFFER
        MOVE.L  A0,VAR(A2)
        ....

VAR     EQU *-DTTX+4
        DC.L    0
        ....

        OFFSET *-DTTX+4
CNT     DC.W    0
BUF     DS.B    10
```

E.2 DRIVER REGISTER USAGE

The PDOS file manager passes all parameters in registers to I/O drivers. All registers are available for use by the driver except address registers A4 through A7.

Parameters in registers

Preserve registers A4 through A7

The driver executes in supervisor mode. The return address is already on the system stack. The status register passes the error conditions back to the PDOS file manager. An 'EQ' status indicates that no error occurred. A 'NE' status specifies an error with the error number returned in data register D0.

Supervisor mode

Status register returns driver results

EQ = ok
NE = Error, D0=error #

The data and address registers of the file manager call are located on the stack immediately following the return address, where D0 is 4(A7), D1 is 8(A7), and so on. This is useful for passing the number of bytes on the end of file to the D3.L of the file manager call. See the input driver example E.6.

If the driver alters constants within the buffer, then the file altered bit must be set in the file slot so that the buffer is correctly restored when rolled to the disk. This is done by executing the instruction 'ORI.W #$8000,12(A4)' or 'TAS.B 12 (A4)'.

Driver altered

The following table describes the register usage for each driver entry point:

```
   OPEN:  D7.W = Channel status
          (A2) = Driver base + 4
          (A4) = File slot
          (A5) = SYSRAM
          (A6) = Task TCB
          (A7) = Return address

  CLOSE:  D7.W = Channel status
          (A2) = Driver base + 4
          (A4) = File slot
          (A5) = SYSRAM
          (A6) = Task TCB
          (A7) = Return address

   READ:  D5.L = Character count (-1 = Line operation)
          D7.W = Channel status
          (A2) = Driver base + 4
          (A3) = Memory buffer
          (A4) = File slot
          (A5) = SYSRAM
          (A6) = Task TCB
          (A7) = Return address
  3*4+4 (A7) = Return EOF bytes to D3.L
```

(E.2 DRIVER REGISTER USAGE continued)

```
   WRITE:  D5.L = Character count (-1 = Line operation)
           D7.W = Channel status
           (A2) = Driver base + 4
           (A3) = Memory buffer
           (A4) = File slot
           (A5) = SYSRAM
           (A6) = Task TCB
           (A7) = Return address

POSITION:  D5.L = Character position
           D7.W = Channel status
           (A2) = Driver base + 4
           (A4) = File slot
           (A5) = SYSRAM
           (A6) = Task TCB
           (A7) = Return address
```

E.3 DRIVER GENERATION

A PDOS driver is generated using conventional PDOS utilities. The procedure is as follows:

1) Assemble the source file using MASM assembler.

2) Change the old driver file type to 'SY' (if defined).

3) Use the MSYFL utility to create a binary image. The section 0 length (E tag) must not exceed $00FC.

4) Set the new driver file type as 'DR'.

```
>MASM TTO:SR,TTO:RB
68K PDOS Assembler R3.2
ERII, Copyright 1986
SRC=TTO:SR
OBJ=TTO:RB
LST=
ERR=
XRF=
END OF PASS 1
END OF PASS 2
>SA TTO,SY
>MSYFL TTO:RB,TTO
68K PDOS SY File Maker Utility 10/27/83
  Source file = TTO:RB
  Destination File = TTO
   SECTION LENGTH = E00000000CA
    Entry Address = 00000000
>SA TTO,DR
>CF LIST,TTO
```

E.4 RESTRICTIONS

The following summarizes the restrictions when adding an I/O driver to PDOS:

1) Drivers must be written in self-relocating, address independent 68000 assembly language.

2) The driver identification constant $A55A must be the first word of the driver.

3) Driver entry points must immediately follow the driver identification word.

4) An I/O driver code and variables cannot exceed the sector size less four link bytes. This results in a maximum length of 252 bytes.

5) A driver MUST NOT make any console or file I/O system calls.

6) A driver is exited via an 'RTS' instruction. A 'NE' status condition indicates a driver error with data register D0 passing the error number.

7) Larger drivers can be written, but the excess code must be located elsewhere in memory. See E.7 for an extended driver example.

8) Drivers execute in supervisor mode.

9) Address registers A4, A5, A6, and A7 must be preserved.

E.5 PDOS OUTPUT DRIVER EXAMPLE

The following program is an example of a PDOS I/O driver. The output is to the logical port number found in the TCB variable U1P$.

```
TTO:SR - 68K PDOS TTO DRIVER               68020 PDOS Assembler 10-Dec-86
PAGE: 1         14:44 17-Dec-86            FILE: TTO:SR,WINI #2


              *       TTO:SR          10/03/86
              *************************************************************************
              *                                                                        *
              *              66   888  K   K    PPPP  DDDD   000   SSS                 *
              *             6    8   8 K  K     P   P  D  D O   O S   S                *
              *            6     8   8 K K      P   P  D  D O   O S                    *
              *            6666   888  KK       PPPP   D  D O   O  SSS                 *
              *            6   6 8   8 K K      P      D  D O   O     S                *
              *            6   6 8   8 K  K     P      D  D O   O S   S                *
              *             666   888  K   K    P     DDDD   000   SSS                 *
              *                                                                        *
              *       TTTTT TTTTT  000     DDDD  RRRR  III V   V EEEEE RRRR            *
              *         T     T   O   O     D  D R   R  I  V   V E     R   R           *
              *         T     T   O   O     D  D R   R  I  V   V E     R   R           *
              *         T     T   O   O     D  D RRRR   I   V V  EEEE  RRRR            *
              *         T     T   O   O     D  D R R    I   V V  E     R R             *
              *         T     T   O   O     D  D R  R   I    V   E     R  R            *
              *         T     T    000     DDDD  R   R III   V   EEEEE R   R           *
              *                                                                        *
              *=************************************************************************
              *       Eyring Research Inst.  Copyright 1983,1986.
              *       ALL RIGHTS RESERVED.
              *=
              *=           Module Name: TTO
              *=                Author: Paul Roper
              *=      Revision History:
              *=
              *=      02/11/86 2.0    Fixed XON/XOFF look before calling put
              *=      06/20/86 3.0    Fixed for upper D1.L=output event # for printers
              *=
0/00000000:   TTO     IDNT    3.0     68K PDOS TTO DRIVER
              *=
```

(E.5 PDOS OUTPUT DRIVER EXAMPLE continued)

```
                                *=*************************************************************************
                                *
                                *       This driver is intended to output files to the terminal.  It outputs
                                *       the file data to the Unit 1 Port (U1P$) of the task that opened
                                *       it.  It filters the output stream by ignoring <LF>, converting
                                *       <CR> characters to <CR><LF> pairs, keeping an independent column
                                *       counter and expanding <TAB> to column positions (multiples of 8),
                                *       using blanks.  <BS> backspace characters decrement the counter.
                                *       Output events, XON/XOFF, and DTR line checks are all supported.
                                *
                                *       D5.L = Character count (-1 = Line)
                                *       D7.W = Channel status
                                *       (A2) = Driver base + 4
                                *       (A3) = Memory buffer
                                *       (A4) = File slot
                                *       (A5) = SYSRAM
                                *       (A6) = Task TCB
                                *       (A7) = Return address
                                *
                      00001400          OPT     PDOS,CRE
                      0000001E BURT     EQU     $001E           ;BIOS UART TBL
                                *
0/00000000:           0/00000000        SECTION 0
```

(E.5 PDOS OUTPUT DRIVER EXAMPLE continued)

```
TTO:SR - 68K PDOS TTO DRIVER             68020 PDOS Assembler 10-Dec-86
PAGE: 2        14:44 17-Dec-86           FILE: TTO:SR,WINI #2

0/00000000:A55A              DTTO   DC.W    $A55A            ;DRIVER ID
0/00000002:600E              DROP   BRA.S   OPEN             ; 2 OPEN
0/00000004:6050              DRCL   BRA.S   CLOS             ; 4 CLOSE
0/00000006:6006              DRRD   BRA.S   READ             ; 6 READ
0/00000008:6050              DRWR   BRA.S   WRIT             ; 8 WRITE
0/0000000A:7046              DRPS   MOVEQ.L #70,D0           ;10 POSITION ERROR
0/0000000C:4E75                     RTS
                             *
0/0000000E:7050              READ   MOVEQ.L #80,D0           ;ERROR 80, DRIVER ERROR
0/00000010:4E75                     RTS
                             *
0/00000012:006C8000000C      OPEN   ORI.W   #$8000,12(A4)    ;FILE ALTERED
0/00000018:422A00EA                 CLR.B   CCNT(A2)         ;CLEAR COUNTER
0/0000001C:4241                     CLR.W   D1               ;D1=PORT #
0/0000001E:122E0452                 MOVE.B  U1P$(A6),D1      ;D1=PORT #
0/00000022:7650                     MOVEQ.L #80,D3
0/00000024:D601                     ADD.B   D1,D3
0/00000026:354300E8                 MOVE.W  D3,OUTE(A2)      ;D3=OUTPUT EVENT #
0/0000002A:16351058                 MOVE.B  UTYP.(A5,D1.W),D3 ;D3=UART TYPE
0/0000002E:154300EB                 MOVE.B  D3,TYPE(A2)      ;SAVE FOR FUTURE
0/00000032:D643                     ADD.W   D3,D3            ;POINT TO DSR
0/00000034:2055                     MOVEA.L (A5),A0
0/00000036:D0F0301E                 ADDA.W  BURT(A0,D3.W),A0
0/0000003A:5448                     ADDQ.W  #2,A0            ;A0=PUTC ENTRY
0/0000003C:254800D0                 MOVE.L  A0,PUTC(A2)      ;SAVE PUTC ADR
0/00000040:E549                     LSL.W   #2,D1            ;SAVE BASE ADR
0/00000042:41ED0158                 LEA.L   UART.(A5),A0
0/00000046:2570100000E0             MOVE.L  0(A0,D1.W),PADR(A2)
0/0000004C:E449                     LSR.W   #2,D1            ;SAVE FLAGS
0/0000004E:48751048                 PEA.1   F8BT.(A5,D1.W)   ;PUSH POINTER TO FLAGS
0/00000052:255F00E4                 MOVE.L  (A7)+,FADR(A2)   ;SAVE PTR
                             *
0/00000056:4240              CLOS   CLR.W   D0               ;RETURN .EQ.
0/00000058:4E75                     RTS
                             *
```

(E.5 PDOS OUTPUT DRIVER EXAMPLE continued)

```
                                *******************************************
                                *       WRITE CHARACTERS
                                *
0/0000005A:006C8000000C         WRIT    ORI.W   #$8000,12(A4)   ;N, ALTERED
                                *
0/00000060:7000                 WRIT02  MOVEQ.L #0,D0           ;GET CHARACTER
0/00000062:101B                         MOVE.B  (A3)+,D0        ;DONE?
0/00000064:6604                           BNE.S WRIT04          ;N
0/00000066:4A85                         TST.L   D5              ;Y, WRITE LINE?
0/00000068:6BEC                           BMI.S CLOS            ;Y, DONE
                                *
0/0000006A:0C000008             WRIT04  CMPI.B  #$08,D0         ;BACKSPACE?
0/0000006E:6604                           BNE.S WRIT06          ;N
0/00000070:532A00EA                     SUBQ.B  #1,CCNT(A2)     ;Y
                                *
0/00000074:0C000009             WRIT06  CMPI.B  #$09,D0         ;OK, TAB?
0/00000078:6614                           BNE.S WRIT08          ;N
0/0000007A:7020                         MOVEQ.L #' ',D0         ;Y
0/0000007C:7207                         MOVEQ.L #7,D1           ;GET MASK
0/0000007E:C22A00EA                     AND.B   CCNT(A2),D1     ;GET COUNTER
0/00000082:5F01                         SUBQ.B  #7,D1           ;TAB BOUNDARY?
```

(E.5 PDOS OUTPUT DRIVER EXAMPLE continued)

```
TTO:SR - 68K PDOS TTO DRIVER                68020 PDOS Assembler 10-Dec-86
PAGE: 3        14:44 17-Dec-86              FILE: TTO:SR,WINI #2

0/00000084:6708                        BEQ.S WRIT08         ;Y
0/00000086:534B                       SUBQ.W  #1,A3         ;N, DO AGAIN
0/00000088:4A85                       TST.L   D5            ;WRITE LINE?
0/0000008A:6B02                        BMI.S WRIT08         ;Y
0/0000008C:5285                       ADDQ.L  #1,D5         ;N, BACKUP
                               *
0/0000008E:0C00000A            WRIT08 CMPI.B  #$0A,D0       ;LF?
0/00000092:6742                        BEQ.S WRIT16         ;Y, IGNORE
0/00000094:0C00000D                   CMPI.B  #$0D,D0       ;N, CR?
0/00000098:6608                        BNE.S WRIT10         ;N
0/0000009A:422A00EA                   CLR.B   CCNT(A2)      ;Y, CLEAR CCNT
0/0000009E:303C0A0D                   MOVE.W  #$0A0D,D0     ;CHANGE TO CRLF
                               *
0/000000A2:0C000020            WRIT10 CMPI.B  #' ',D0       ;CONTROL?
0/000000A6:6D04                        BLT.S WRIT12         ;Y
0/000000A8:522A00EA                   ADDQ.B  #1,CCNT(A2)   ;N, UP COUNT
                               *
0/000000AC:4A2A00EB            WRIT12 TST.B   TYPE(A2)      ;DEFINED TYPE?
0/000000B0:67A4                        BEQ.S CLOS           ;N, SKIP IT
0/000000B2:222A00E8                   MOVE.L  OUTE(A2),D1   ;GET OUT EFVENT TO UPPER WORD OF D1
0/000000B6:206A00E4                   MOVEA.L FADR(A2),A0   ;GET PTR TO FLGS
0/000000BA:1210                       MOVE.B  (A0),D1       ;TEST FLAG EACH TIME
0/000000BC:08010000                   BTST.L  #0,D1         ;^S^Q CHECK?
0/000000C0:6704                        BEQ.S WRIT14         ;N
0/000000C2:4A01                       TST.B   D1            ;Y, ^S STOP SET?
0/000000C4:6BE6                        BMI.S WRIT12         ;Y, WAIT HERE
                               *
0/000000C6:206A00E0            WRIT14 MOVEA.L PADR(A2),A0   ;UART BASE ADR
0/000000CA:4EB900000000               DC.W    $4EB9,0,0     ;JSR PUTC.L
                      000000D0 PUTC   EQU     *-DTTO        ;RETRY?
0/000000D0:66DA                        BNE.S WRIT12         ;Y
0/000000D2:E048                       LSR.W   #8,D0         ;N, 2 CHARS?
0/000000D4:66D6                        BNE.S WRIT12         ;Y
                               *
0/000000D6:5385                WRIT16 SUBQ.L  #1,D5         ;DONE?
0/000000D8:6686                        BNE.S WRIT02         ;N
                               *      BRA     CLOS2         ;Y
0/000000DA:4E75                       RTS                   ;Y, RETURN .EQ.
                               *
```

(E.5 PDOS OUTPUT DRIVER EXAMPLE continued)

```
                                      ******************************************
                                      *       DRIVER VARIABLES
                                      *
0/000000DC:          000000E0                 OFFSET  *-DTTO+4
  000000E0:00000000                   PADR    DC.L    0              ;BASE ADR
  000000E4:00000000                   FADR    DC.L    0              ;UART FLAGS ADDRESS
  000000E8:0000                       OUTE    DC.W    0              ;OUTPUT EVENT #
  000000EA:00                         CCNT    DC.B    0              ;COLUMN COUNT
  000000EB:00                         TYPE    DC.B    0              ;PORT TYPE
  000000EC:                                   EVEN
                                      *
                                      ******************************************
                                      *       DRIVER LENGTH CHECK
                                      *
  000000EC:                                   IFLT    256-(TYPE+1)
                                              FAIL    ** DRIVER LENGTH ERROR! **
                                              ENDC
```

```
TTO:SR - 68K PDOS TTO DRIVER          68020 PDOS Assembler 10-Dec-86
PAGE: 4        14:44 17-Dec-86        FILE: TTO:SR,WINI #2
```

```
                                      *
  000000EC:      0/00000000                   END     DTTO
```

(E.5 PDOS OUTPUT DRIVER EXAMPLE continued)

```
TTO:SR - 68K PDOS TTO DRIVER          68020 PDOS Assembler 10-Dec-86
PAGE: 5        14:44 17-Dec-86        FILE: TTO:SR,WINI #2


SYMBOL CROSS REFERENCE:

BURT      E     0000001E       *1/54   2/23
CCNT             000000EA      2/13    2/49    2/55    3/11    3/16    *3/47
CLOS            0/00000056     2/3     *2/33   2/45    3/19
DRCL      R     0/00000004     *2/3
DROP      R     0/00000002     *2/2
DRPS      R     0/0000000A     *2/6
DRRD      R     0/00000006     *2/4
DRWR      R     0/00000008     *2/5
DTTO            0/00000000     *1/1    3/30    3/43    4/2
F8BT.     E      00000048      2/30
FADR             000000E4      2/31    3/21    *3/45
OPEN            0/00000012     2/2     *2/12
OUTE             000000E8      2/18    3/20    *3/46
PADR             000000E0      2/28    3/28    *3/44
PUTC      E      000000D0      2/25    *3/30
READ            0/0000000E     2/4     *2/9
TTO       R     0/00000000     *1/32
TYPE             000000EB      2/20    3/18    *3/48   3/54
U1P$      E      00000452      2/15
UART.     E      00000158      2/27
UTYP.     E      00000058      2/19
WRIT            0/0000005A     2/5     *2/39
WRIT02          0/00000060     *2/41   3/36
WRIT04          0/0000006A     2/43    *2/47
WRIT06          0/00000074     2/48    *2/51
WRIT08          0/0000008E     2/52    2/1     3/4     *3/7
WRIT10          0/000000A2     3/10    *3/14
WRIT12          0/000000AC     3/15    *3/18   3/26    3/31    3/33
WRIT14          0/000000C6     3/24    *3/28
WRIT16          0/000000D6     3/8     *3/35

UNREFERENCED SYMBOLS:

DRCL      R   0/00000004   DROP     R   0/00000002   DRPS     R   0/0000000A
DRRD      R   0/00000006   DRWR     R   0/00000008   TTO      R   0/00000000
```

E.6 PDOS INPUT DRIVER EXAMPLE

```
TTI:SR - 68K PDOS TTI DRIVER            68020 PDOS Assembler 10-Dec-86
PAGE: 1          14:45 17-Dec-86        FILE: TTI:SR,WINI #2


                                *       TTI:SR          10/06/86
                                *************************************************************************
                                *                                                                       *
                                *              66   888  K   K    PPPP  DDDD   000   SSS                *
                                *             6    8   8 K  K     P   P  D  D O   O S   S               *
                                *            6     8   8 K K      P   P  D  D O   O S                   *
                                *            6666   888  KK       PPPP   D  D O   O  SSS                *
                                *            6   6 8   8 K K      P      D  D O   O     S               *
                                *            6   6 8   8 K  K     P      D  D O   O S   S               *
                                *             666   888  K   K    P     DDDD   000   SSS                *
                                *                                                                       *
                                *        TTTTT TTTTT III    DDDD  RRRR  III V   V EEEEE RRRR            *
                                *          T     T    I      D  D R   R  I  V   V E     R   R           *
                                *          T     T    I      D  D R   R  I  V   V E     R   R           *
                                *          T     T    I      D  D RRRR   I   V V  EEEE  RRRR            *
                                *          T     T    I      D  D R R    I   V V  E     R R             *
                                *          T     T    I      D  D R  R   I    V   E     R  R            *
                                *          T     T   III    DDDD  R   R III   V   EEEEE R   R           *
                                *                                                                       *
                                *=***********************************************************************
                                *       Eyring Research Inst.  Copyright 1983,1986.
                                *       ALL RIGHTS RESERVED.
                                *=
                                *=           Module Name: TTI
                                *=                Author: Richard Adams
                                *=      Revision History:
                                *=
                                *=      10/03/86 3.0    Initial release
                                *=
0/00000000:                     TTI     IDNT    3.0     68K PDOS TTI DRIVER
                                *=
                                *=***********************************************************************
```

(E.6 PDOS INPUT DRIVER EXAMPLE continued)

```
                                   *
                                   * This driver is intended to input files from the terminal.  It gets
                                   * characters from the input port (PRT$) of the task that opened it,
                                   * stores them in the buffer (A3), and echoes them to active ouput port(s).
                                   * It supports both XRLF read line and XRBF read block primitives.  OPEN
                                   * call simply makes sure that there is an input port assigned to the task.
                                   * Close does nothing.  EOF errors are returned, along with the byte count,
                                   * if an escape is entered.
                                   *
                                   *      D5.L = Character count (-1 = Line)
                                   *      D7.W = Channel status
                                   *      (A2) = Driver base + 4
                                   *      (A3) = Memory buffer
                                   *      (A4) = File slot
                                   *      (A5) = SYSRAM
                                   *      (A6) = Task TCB
                                   *      (A7) = Return address
                                   *
                       00001000           OPT     PDOS
                                   *
0/00000000:          0/00000000           SECTION 0
0/00000000:A55A                    DTTI   DC.W    $A55A          ;DRIVER ID
0/00000002:600E                    DROP   BRA.S   OPEN           ; 2 OPEN
```

(E.6 PDOS INPUT DRIVER EXAMPLE continued)

```
TTI:SR - 68K PDOS TTI DRIVER          68020 PDOS Assembler 10-Dec-86
PAGE: 2        14:45 17-Dec-86        FILE: TTI:SR,WINI #2

0/00000004:6012          DRCL    BRA.S   CLOS            ; 4 CLOSE
0/00000006:6014          DRRD    BRA.S   READ            ; 6 READ
0/00000008:6004          DRWR    BRA.S   WRIT            ; 8 WRITE
0/0000000A:7046          DRPS    MOVEQ.L #70,D0          ;10 POSITION ERROR
0/0000000C:4E75                  RTS
                         *
0/0000000E:7050          WRIT    MOVEQ.L #80,D0          ;ERROR 80, DRIVER ERROR
0/00000010:4E75                  RTS
                         *
0/00000012:4A2E044F      OPEN    TST.B   PRT$(A6)        ;IS THERE INPUT PORT?
0/00000016:67F6                    BEQ.S WRIT            ;N, SEND ERROR 80
                         *
0/00000018:4240          CLOS    CLR.W   D0              ;RETURN .EQ.
0/0000001A:4E75                  RTS
                         *
                         ******************************************
                         *       READ CHARACTERS, BLOCK OR LINE
                         *
0/0000001C:7200          READ    MOVEQ.L #0,D1           ;GET COUNT, EOF FOR ECSAPE
                         *
                         *       DO LINE/BLOCK READ
                         *
0/0000001E:A07A          LINE    XGCR                    ;GET A CHARACTER
0/00000020:6D1E                    BLT.S ESC             ;ESCAPE OUT
0/00000022:4A85                  TST.L   D5              ;LINE?
0/00000024:6A0A                    BPL.S @010            ;N, SKIP [CR] CHECK
0/00000026:0C00000D              CMPI.B  #13,D0          ;Y, CR?
0/0000002A:6604                    BNE.S @010            ;N, ECHO AND STORE
0/0000002C:4213                  CLR.B   (A3)            ;Y, TERMINATE LINE
0/0000002E:60E8                  BRA     CLOS            ;GET BAT OUT
                         *
0/00000030:A086          @010    XPCC                    ;ECHO TO SCREEN
0/00000032:16C0                  MOVE.B  D0,(A3)+        ;SAVE IN BUFFER
0/00000034:5281                  ADDQ.L  #1,D1           ;UP COUNT
0/00000036:4A85                  TST.L   D5              ;LINE?
0/00000038:6BE4                    BMI.S LINE            ;Y, SKIP COUNT CHECK
0/0000003A:B285                  CMP.L   D5,D1           ;N, DONE BLOCK COUNT?
0/0000003C:6DE0                    BLT.S LINE            ;N, GET ANOTHER
0/0000003E:60D8                  BRA.S   CLOS            ;Y, RETURN .EQ.
                         *
0/00000040:2F410010      ESC     MOVE.L  D1,3*4+4(A7)    ;RETURN COUNT IN OLD D3
0/00000044:7038                  MOVEQ.L #56,D0          ;EOF ERROR RETURN
0/00000046:4E75                  RTS
                         *
```

(E.6 PDOS INPUT DRIVER EXAMPLE continued)

```
                                 ******************************************
                                 *       DRIVER LENGTH CHECK
                                 *
0/00000048:                              IFLT    256-(*-DTTI+4)
                                         FAIL    ** DRIVER LENGTH ERROR! **
                                         ENDC
                                 *
0/00000048:      0/00000000              END     DTTI


TTI:SR - 68K PDOS TTI DRIVER             68020 PDOS Assembler 10-Dec-86
PAGE: 3          14:45 17-Dec-86         FILE: TTI:SR,WINI #2


DEFINED SYMBOLS:

CLOS         0/00000018   DRCL     R   0/00000004   DROP     R   0/00000002
DRPS     R   0/0000000A   DRRD     R   0/00000006   DRWR     R   0/00000008
DTTI         0/00000000   ESC          0/00000040   LINE         0/0000001E
OPEN         0/00000012   PRT$     E     0000044F   READ         0/0000001C
TTI      R   0/00000000   WRIT         0/0000000E

EXTERNAL DEFINITIONS: NONE

EXTERNAL REFERENCES: NONE

UNDEFINED SYMBOLS: NONE

UNREFERENCED SYMBOLS:

DRCL     R   0/00000004   DROP     R   0/00000002   DRPS     R   0/0000000A
DRRD     R   0/00000006   DRWR     R   0/00000008   TTI      R   0/00000000
```

E.7 EXTENDED DRIVER EXAMPLE

PDOS I/O drivers must reside in the channel buffer, which is only 256 bytes long. The forward and backward file links take 4 bytes and the dedicated BRA.S table takes 6*2 more bytes, leaving only 240 bytes (=256-4-12) to work with. You can expand I/O drivers beyond this limit, by having code resident with PDOS.

The following working example shows a multiple expanded driver file called EXT:SR. The idea is that you add as many large drivers as you want to the xxBIOS:SR file for your system, using the structure described below. Then to access them, you create some new disk resident drivers from the EXT:SR file, differentiating them by DNUM=0,1,2,...

For example, to create files to access extended drivers #0 and #1 you would do the following:

```
0>SA DRV0,SY
0>MASM EXT:SR/DNUM=0,#DRV0
0>MSYFL DRV0,DRV0
0>SA DRV0,DR
0>SA DRV1,SY
0>MASM EXT:SR/DNUM=1,#DRV1
0>MSYFL DRV1,DRV1
0>SA DRV1,DR
0>_
```

Now there are two drivers, DRV0 and DRV1, to access each extended driver #0 and #1. This EXT:SR driver is a fixed length, which is important if you are going to store variables within the driver channel.

(E.7 EXTENDED DRIVER EXAMPLE continued)

The only interesting call to EXT is OPEN, when it looks for the R$TASK table and a special EXT driver ID word ($5AA5). If you don't have any expanded driver code in the BIOS you booted, then EXT returns all calls with an error #99, but will not crash your system. If EXT finds the ID word, then it stores the address of the specified BRA.L instruction IN THE DRIVER at $10(A2). All the other entries to EXT just load up D0.L with the driver # (0,2,4,...) and an entry offset (0=open 4=close, 8=read,...) before branching (with an RTS) into the BIOS extended code entry point (stored in $10(A2)).

This keeps things all position independent, relocatable and re-entrant. Let's look at the EXT code before diving into the BIOS:

```
        TTL     EXT:SR - 68K PDOS 68K PDOS EXT DRIVER
*       EXT:SR          06/27/86
*************************************************************************
*                                                                       *
*            66   888  K   K    PPPP  DDDD   OOO   SSS                  *
*           6    8   8 K  K     P   P D   D O   O S   S                 *
*          6     8   8 K K      P   P D   D O   O S                     *
*          6666   888  KK       PPPP  D   D O   O  SSS                  *
*          6   6 8   8 K K      P     D   D O   O     S                 *
*          6   6 8   8 K  K     P     D   D O   O S   S                 *
*           666   888  K   K    P     DDDD   OOO   SSS                  *
*                                                                       *
*       EEEEE X   X TTTTT    DDDD  RRRR  III V   V EEEEE RRRR           *
*       E      X X    T      D   D R   R  I  V   V E     R   R          *
*       E       X     T      D   D R   R  I  V   V E     R   R          *
*       EEEE    X     T      D   D RRRR   I   V V  EEEE  RRRR           *
*       E      X X    T      D   D R R    I   V V  E     R R            *
*       E     X   X   T      D   D R  R   I    V   E     R  R           *
*       EEEEE X   X   T      DDDD  R   R III   V   EEEEE R   R          *
*                                                                       *
*=***********************************************************************
*       Eyring Research Inst.  Copyright 1983,1986.
*       ALL RIGHTS RESERVED.
*=
*=          Module Name: EXT
*=               Author: Richard Adams
*=      Revision History:
*=
*=      06/27/86 3.0    Initial version of extended driver
*=
EXT     IDNT    3.0     68K PDOS EXT DRIVER
*=
*=**********************************************************
```

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
*
*       This driver is a general extended I/O driver, that
*       can be adapted for expanded driver code over the
*       252 byte limit.
*
*       D5.L = Character count (-1 = Line)
*       D7.W = Channel status
*       (A2) = Driver base + 4
*       (A3) = Memory buffer
*       (A4) = File slot
*       (A5) = SYSRAM
*       (A6) = Task TCB
*       (A7) = Return address
*
        IFUDF   DNUM    :DNUM   EQU     0       ;DEFAULT TO DRIVER #0
        PRINT   '  ** Extended driver # ',DNUM
        IFGT    DNUM-5
        PRINT   '  ** ERROR, Driver numbers only 0-5'
        ENDC
        PAGE
        SECTION 0

DEXT    DC.W    $A55A           ;DRIVER ID
DROP    BRA.S   OPEN            ; 2 OPEN
DRCL    BRA.S   CLOS            ; 4 CLOSE
DRRD    BRA.S   READ            ; 6 READ
DRWR    BRA.S   WRIT            ; 8 WRITE
DRPS    BRA.S   POSI            ;10 POSITION
        DC.L    0               ;Location of expanded code in BIOS
CODE    EQU     $10             ;CODE is channel offset of this saver
*
OPEN    ORI.W   #$8000,12(A4)   ;FILE ALTERED
        MOVEA.L (A5),A1         ;GET ADDR OF B$BIOS
        ADDA.L  (A1),A1         ;GET ADDRESS OF R$TASK TABLE
        CMPI.W  #$5AA5,-(A1)    ;IS ID THERE?
          BNE.S ERROR           ;N, DRIVER ERROR
        SUBQ.W  #4,A1           ;Y, POINT TO XTENDED CODE 'BRA.L'
        MOVE.L  A1,CODE(A2)     ;SAVE ENTRY
*
        MOVEQ.L #0,D0           ;0=open
```

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
*
*       CALL EXTENDED CODE WITH ENTRY OFFSET:
*         D0.L = <minor offset> | <major offset>
*
*          Where <major offset> = 0 driver #0
*                               = 2 driver #1
*                               = 4 driver #2, etc.
*
*          Where <minor offset> = 0 open
*                               = 4 close
*                               = 8 read
*                               =12 write
*                               =16 position
*
CALL    MOVE.L  CODE(A2),-(A7)  ;GET ADDRESS
          BEQ.S EXTER           ;NO CODE, RETURN .NE.
        SWAP    D0
        MOVE.W  #DNUM*2,D0      ;GET DRIVER NUMBER OFFSET
        RTS                     ;GO TO CODE IN BIOS
*
CLOS    MOVEQ.L #4,D0           ;4=close
        BRA.S   CALL
*
READ    MOVEQ.L #8,D0           ;8=read
        BRA.S   CALL
*
WRIT    MOVEQ.L #12,D0          ;12=write
        BRA.S   CALL
*
POSI    MOVEQ.L #16,D0          ;16=position
        BRA.S   CALL
*
EXTER   ADDQ.W  #4,A7           ;POP CODE ADDRESS
*
ERROR   MOVEQ.L #99,D0          ;if no extended driver code, err 99
        RTS
        END     DEXT
```

Note that from SYRAM (A5), you get the address of B$BIOS table and then calculate the address of R$TASK table. Place your $5AA5 EXT ID word right before R$TASK and the 'BRA.L XCODE' right before that.

To look at the xxBIOS:SR changes that let you add code there, let's get the example. The EXT example uses the TTA driver, adding it to the MVME117 V7BIOS:SR file. Just before the R$TASK table in the xxBIOS:SR file, you insert a BRA.L XCODE and an $5AA5 data word, as follows:

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
        ...
B$STRT  BRA.L   BSTRT           ;BOOT EPROM START
        DC.L    PDID            ;PDOS BOOT IDENTIFICATION
        DC.W    SYID            ;SYSTEM ID
B.SRAM  DC.L    S$SRAM          ;SYRAM ADDRESS
*
        BRA.L   XCODE           ;GO TO DRIVER CODE
        DC.W    $5AA5           ;EXTENDED DRIVER ID WORD
*
*****************************************************
*       TASK STARTUP TABLE (NON-RUN MODULE)
*
        IFEQ    RF
        XDEF    R$TASK
*
R$TASK  DC.B    1,U.1TYP,BR,%0000       ;PORT #1
        ...
```

Now, insert the driver code following the BIOS interrupt routines, but preceding the INCLUDE MBIOS:SR command. This could be done using an INCLUDE command, or even conditionally on an assembly flag. Define NDRV equal to the number of extended drivers in the xxBIOS (NDRV=1 in the example). You then have your major switchboard routine, XCODE, which checks the driver #, returning error 99 if it is too big. If D0.W is in range, then XCODE jumps to the particular driver code called by DRV0,DRV1, etc., with a JMP:

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
         ...
**********************************************************
*        EXTENDED DRIVER MAJOR ENTRY
*         IN:   D0.L = MINOR (0,4,8,12,16) | MAJOR (0,2,4,...)
*
NDRV     EQU     1                   ;NUMBER OF DRIVERS RESIDENT
*
XCODE    CMPI.W  #NDRV*2,D0          ;IS MAJOR BRA.L IN TABLE?
           BLO.S @010                ;Y, GO TO IT
         MOVEQ.L #99,D0              ;N, THEN ILLEGAL
         RTS
*
@010     JMP     MAJOR(PC,D0.W)      ;GO TO DRIVER ENTRY
*
*        Main multiple driver switchboard table has each major
*        device entry is 4 bytes long, for a 'BRA.L DRVx' instruction.
*        The range is checked using NDRV, the number of drivers in BIOS.
*
MAJOR    BRA.L   DRV0                ;DRIVER #0 (TTA)
*        BRA.L   DRV1                ;DRIVER #1
*        BRA.L   DRV2                ;DRIVER #2
*        ...
*
```

In the example, only the standard TTA driver code has been added as DRV0. Since the driver entry points are now 0, 4, 8, 12, 16, you can have long jumps to the driver entry points, not limited to the 128 byte range. Another bonus is that for entries that are to return an error, such as read and position, you can handle the error RIGHT IN THE BRANCH TABLE! This is done by loading the error with a MOVEQ.L and RTS.

Variables within the driver (offset from A2) are very easy to define in the BIOS. Since you know the size of EXT:SR to be $4C, then by taking links into account you just use an OFFSET $50 directive, followed by DS.L, DS.W, and DS.B commands to yield the proper (A2) driver offsets. Remember to exit the OFFSET mode with a SECTION 14 command, for the linker:

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
*****************************************************
*       Extended Driver #0: TTA
*
*       Driver variables go here, starting at (A2) offset = $50
*       Use OFFSET and then return to section 14.
*
        OFFSET  $50             ;end of EXT driver code in buffer
PADR    DS.L    1               ;DC.L BASE ADR
FADR    DS.L    1               ;DC.L UART FLAGS ADDRESS

OUTE    DS.W    1               ;DC.W OUTPUT EVENT #
CCNT    DS.B    1               ;DC.B COLUMN COUNT
TYPE    DS.B    1               ;DC.B PORT TYPE
PUTC    DS.L    1               ;DC.L PUT CHAR ADDRESS FOR JSR
        SECTION 14              ;back to BIOS section
```

The next requirement is to reference in any external offsets or addresses:

```
*
*       Next define and XREF any needed offsets for SYRAM, etc.
*
BURT    EQU     $001E           ;BIOS UART TBL
        XREF    U2P$,UTYP.,UART.,F8BT.
```

Now, go to the specific driver code, which swaps D0 to get the open, close, read, write, or position offset and branches into the fixed entry table to perform the driver function:

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
*       Here is the minor entry switchboard, with JMP offset in
*       upper word of D0.L.  Minor entry offsets are 0,4,8,$C,$10
*       for open, close, read, write and position.  This allows
*       errors in BRA.L table, with sequences like:
*
*               MOVEQ.L #ERR,D0
*               RTS
*
DRV0    SWAP    D0              ;MINOR OFFSET IN D0.W LOWER
        JMP     DRV0TB(PC,D0.W) ;GO TO SPECIFIC MINOR ENTRY...
*
*       DRV0TB  BRA.L   OPEN
*               BRA.L   CLOS
*               BRA.L   READ
*               BRA.L   WRIT
*               BRA.L   POSIT
*
DRV0TB  BRA.L   OPEN            ;0=OPEN
*
        BRA.L   CLOS            ;4=CLOSE
*
        MOVEQ.L #80,D0          ;8=READ: ERROR 80, DRIVER ERROR
        RTS
*
        BRA.L   WRIT            ;12=WRITE
*
        MOVEQ.L #70,D0          ;16=POSITION: ERROR 70, POSITION ERR
        RTS
*
OPEN    ORI.W   #$8000,12(A4)   ;FILE ALTERED
        CLR.B   CCNT(A2)        ;CLEAR COUNTER
        CLR.W   D1              ;D1=PORT #
        MOVE.B  U2P$(A6),D1     ;D1=PORT #
        MOVEQ.L #80,D3
        ADD.B   D1,D3
        MOVE.W  D3,OUTE(A2)     ;D3=OUTPUT EVENT #
        MOVE.B  UTYP.(A5,D1.W),D3 ;D3=UART TYPE
        MOVE.B  D3,TYPE(A2)     ;SAVE FOR FUTURE
        ADD.W   D3,D3           ;POINT TO DSR
        MOVEA.L (A5),A0
        ADDA.W  BURT(A0,D3.W),A0
        ADDQ.W  #2,A0           ;A0=PUTC ENTRY
        MOVE.L  A0,PUTC(A2)     ;SAVE PUTC ADR
        LSL.W   #2,D1           ;SAVE BASE ADR
        LEA.L   UART.(A5),A0
        MOVE.L  0(A0,D1.W),PADR(A2)
        LSR.W   #2,D1           ;SAVE FLAGS
        PEA     F8BT.(A5,D1.W)  ;PUSH POINTER TO FLAGS
        MOVE.L  (A7)+,FADR(A2)  ;SAVE PTR
        BRA.S   CLOS2
```

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
*
CLOS    MOVEQ.L #$0C,D0         ;GET FF
        MOVEQ.L #1,D5           ;DO 1 CHAR
        BRA.S   WRIT12          ;OUT IT
*
CLOS2   CLR.W   D0              ;RETURN .EQ.
        RTS
*
******************************************************
*       WRITE CHARACTERS
*
WRIT    ORI.W   #$8000,12(A4)   ;N, ALTERED
*
WRIT02  MOVEQ.L #0,D0           ;GET CHARACTER
        MOVE.B  (A3)+,D0        ;DONE?
          BNE.S WRIT04          ;N
        TST.L   D5              ;Y, WRITE LINE?
          BMI.S CLOS2           ;Y, DONE
*
WRIT04  CMPI.B  #$08,D0         ;BACKSPACE?
          BNE.S WRIT06          ;N
        SUBQ.B  #1,CCNT(A2)     ;Y
*
WRIT06  CMPI.B  #$09,D0         ;OK, TAB?
          BNE.S WRIT08          ;N
        MOVEQ.L #' ',D0         ;Y
        MOVE.B  CCNT(A2),D1     ;GET COUNTER
        LSL.B   #5,D1           ;$CCC0 0000
        CMPI.B  #7<<5,D1        ;TAB BOUNDARY?
          BEQ.S WRIT08          ;Y
        SUBQ.W  #1,A3           ;N, DO AGAIN
        TST.L   D5              ;WRITE LINE?
          BMI.S WRIT08          ;Y
        ADDQ.L  #1,D5           ;N, BACKUP
*
WRIT08  CMPI.B  #$0A,D0         ;LF?
          BEQ.S WRIT16          ;Y, IGNORE
        CMPI.B  #$0D,D0         ;N, CR?
          BNE.S WRIT10          ;N
        CLR.B   CCNT(A2)        ;Y, CLEAR CCNT
        MOVE.W  #$0A0D,D0       ;CHANGE TO CRLF
*
WRIT10  CMPI.B  #' ',D0         ;CONTROL?
          BLT.S WRIT12          ;Y
        ADDQ.B  #1,CCNT(A2)     ;N, UP COUNT
```

(E.7 EXTENDED DRIVER EXAMPLE continued)

```
*
WRIT12  TST.B    TYPE(A2)       ;DEFINED TYPE?
          BEQ.S CLOS2           ;N, SKIP IT
        MOVE.L   OUTE(A2),D1    ;GET OUT EFVENT TO UPPER WORD OF D1
        MOVEA.L FADR(A2),A0     ;GET PTR TO FLGS
        MOVE.B   (A0),D1        ;TEST FLAG EACH TIME
        BTST.L   #0,D1          ;^S^Q CHECK?
          BEQ.S WRIT14          ;N
        TST.B    D1             ;Y, ^S STOP SET?
          BMI.S WRIT12          ;Y, WAIT HERE
*
WRIT14  MOVEA.L PADR(A2),A0     ;UART BASE ADR
        MOVEA.L PUTC(A2),A1     ;POINT TO PUTC
        JSR      (A1)           ;CALL PUT CHAR
          BNE.S WRIT12          ;Y
        LSR.W    #8,D0          ;N, 2 CHARS?
          BNE.S WRIT12          ;Y
*
WRIT16  SUBQ.L   #1,D5          ;DONE?
          BNE.S WRIT02          ;N
        RTS                     ;Y, RETURN .EQ.
```

You would add other drivers here, calling them DRV1, DRV2, and so on. If you need more RAM storage than \$100-\$50 (176 bytes), then you would have to handle it separately. Also, you are limited to PDOS booting only up to 255 sectors, or less than 66k bytes for the BIOS, driver code and PDOS. This means that huge drivers must be accommodated differently. Now all that remains is to finish up by including MBIOS:SR.

```
*
        NOL
        PAGE
        INCLUDE MBIOS:SR
        END
```

APPENDIX F

PDOS FLOATING POINT MODULE

Floating point is supported through the 68881 co-processor. The following instructions may be found in the MASM20 PDOS assembler:

68881 co-processor support.

Co-processor default set to 1.

FABS	FMOVECR
FACOS	FMOVEM
FADD	FMUL
FASIN	FNEG
FATAN	FNOP
FATANH	FREM
FBcc	FRESTORE
FCMP	FScc
FCOS	FSAVE
FCOSH	FSCALE
FDBcc	FSGLDIV
FDIV	FSGLMUL
FETOX	FSIN
FETOXM1	FSINCOS
FGETEXP	FSINH
FGETMAN	FSQRT
FINT	FSUB
FINTRZ	FTAN
FLOG10	FTANH
FLOG2	FTENTOX
FLOGN	FTRAPcc
FLOGNP1	FTST
FMOD	FTWOTOX
FMOVE	

DC/DS Floating point Constants

APPENDIX G

GLOSSARY

ASCII Literal ASCII literals are special characters within strings that normally cannot be represented by a single printable character. An ASCII literal is composed of two hex characters within angle brackets.

ASCII Literal

Assembler A language translator that translates ASCII text into machine code.

Assembler

Bad Track Any physical track that contains one or more manufacturing defect(s) which causes either hard or soft data errors in at least one block on the track.

Bad Track

Bad Track Mapping A method of "removing" bad tracks from a disk by skipping the bad tracks when mapping the logical tracks to the physical.

Bad Track Mapping

Bias A logical sector offset used by disk partitions to allow system-dependent data to be stored in the first sectors of a partition. For Winchesters, the bias is usually one track of sectors, and for floppies, it is two tracks (32 sectors, normally).

Bias

BIOS Basic I/O Subsystem. The PDOS BIOS contains the read/write primitives, prompts, map and LED controls, setup paramaters, and other hardware related variables.

BIOS

Bit Map A data structure utilized by PDOS for both memory and file space allocation. A separate bit in the memory bit map is associated with each block of memory in the system. Likewise, each sector on a logical disk device is associated with a single bit in the sector bit map in the disk header. A 'one' indicates the corresponding sector is allocated and a 'zero' indicates that the corresponding sector is free.

Bit Map

(APPENDIX G GLOSSARY continued)

Block	A block is the smallest amount of data that can be requested by PDOS from a controller. The number of bytes per PDOS block is usually 256, but it may also be 512.	Block
Blocked	Another term for the suspended task state.	Blocked
Buffer	A temporary block of memory, usually used for message and I/O transfers.	Buffer
Command Line Interpreter	The Command Line Interpreter is a small system software module which parses a line for commands and parameters. The CLI is called by the PDOS monitor.	Command Line Interpreter
Compiler	A language translator that translates the text of a high level language into assembly or machine code.	Compiler
Concurrency	Processes or tasks whose execution overlaps in time. They may be interacting or independent.	Concurrency
Contention	A situation that occurs when more than one task vies for a single resource.	Contention
Create	A system service that initializes a structure by entering information such as its name, size, etc. into system tables. Specifically, PDOS supports task and file creation.	Create
Critical Code	(Also Critical Section). A portion of software that accesses a shared resource and must be protected so that while one task is performing the access (executing the software), no other task is permitted to access the same resource. In most cases, either interrupts are disabled during the execution of this code or the task is locked.	Critical Code

(APPENDIX G GLOSSARY continued)

Cylinder	That portion of the drive media that is defined by one position of the head assembly. The number of cylinders is the number of stepper positions that the head assembly can read or write data from or to, or that places the head assembly over a data area on the media.	Cylinder
Deadlock	A situation that occurs when one or more tasks within a system are suspended, waiting for resources that have already been assigned to other tasks that are also waiting for additional resources.	Deadlock
Debugger	A system software utility that aids a programmer in locating errors in his software. Functions usually include breakpoints, single stepping, memory inspect and change, disassembly, and assembly.	Debugger
Device	A unit of peripheral hardware such as a printer, terminal, or disk drive.	Device
Device Driver	A system software module that directly controls the data transfer to and from an I/O peripheral. PDOS device drivers are an extension of the file system.	Device Driver
Directory	A data structure containing entries for each file in the file system of a storage device. Each directory entry contains information about the file name, access rights, size, date of creation, and last update.	Directory
Disk	A logical division or portion of a drive, defined and referenced in PDOS with a legal disk number and, possibly, a sector offset. PDOS equates a PDOS disk to either a floppy drive or a Winchester partition. Usage: It makes sense to refer to a "PDOS disk," but it makes no sense to refer to a "Winchester disk."	Disk

(APPENDIX G GLOSSARY continued)

Disk number	A disk number is used by PDOS to reference a disk device. A single hardware device may be referenced by several disk numbers.	Disk number
DMA	An I/O processor memory access technique whereby the system processor is placed in a hold state while the I/O processor transfers data to or from memory, independent of the system processor and usually at the maximum memory data rate.	Direct Memory Access
Drive	A single Winchester or floppy hardware device, usually addressed directly by a controller using a unique device select code (sometimes denotes the LUN for logical unit number). Usage: It makes sense to refer to a "Winchester drive," but it makes no sense to refer to a "PDOS drive."	Drive
Drive Data Block	A RAM data area that contains the parameters, partitions, and bad track list of a drive.	Drive Data Block
Editor	A system utility designed to facilitate the entry and maintenance of text. Typical facilities include file creation, modification, concatenation, or deletion. The PDOS editor is MEDIT, a full-screen editor.	Editor
End-of-File	A soft pointer to the end of "known" data within a file (EOF).	End-of-File
Entry Point	The programmer-defined address at which a task begins executing.	Entry Point
Event	A condition used to synchronize task execution. An event may have a hardware or software origin. Hardware events result from processor interrupts. Software events are either user- or system-defined and are used to coordinate system/user tasks or resources.	Event

(APPENDIX G GLOSSARY continued)

Execution Module	An execution module consists of the PDOS kernel plus other non-file oriented primitives. This object module is linked with user application tasks to form a ROMable, stand-alone program for the target processor. Other execution modules are also linked in for high level language support.	Execution Module
File	A collection of data, normally stored on a storage device such as a disk or tape.	File
File Attributes	File attributes are file status bits indicating the file type, disk storage method, and protection flags.	File Attributes
File Slot	A file slot is a logical I/O channel through which data transfers from a user application to secondary storage or other I/O device. The file slot maintains file status, pointers, and buffers.	File Slot
File System	System software modules that manage files on storage media. Functions include create, delete, rename, read, write, position, protect, etc.	File System
File Type	File type is an attribute used by the PDOS monitor in determining how a file is processed.	File Type
First Fit	An algorithm for memory allocation that searches the free list (bit map) only long enough to find an unused memory block that is large enough to satisfy the memory request.	First Fit
Foreground/ Background	A condition within a multi-tasking operating system where critical programs operate in the foreground and execute with high priority while background assemblies, edits, listings, etc., are also going on at the same or lower priority.	Foreground/Background

(APPENDIX G GLOSSARY continued)

Format	The process of a disk controller which places the ID address marks, the sector header information, the data fields, and gaps onto the drive media. The PDOS format utility not only formats the drive or diskette, but also performs sector interleaving, bad track detection, disk partition definition, and drive parameter definition.	Format
Fragmentation	A condition where main memory or secondary storage is segmented due to dynamic memory allocation and deallocation.	Fragmentation
Friendly Environment	A software environment in which all software is adequately tested and therefore one task does not interfere with or cause errors in the execution of another task. The operating system cannot prevent intertask conflicts.	Friendly Environment
Hard Error	An error which is repeatable.	Hard Error
Head	A device which reads and writes data from and to one surface of a drive. The number of heads is the physical number of data surfaces of a drive, or the number of different head select codes a controller can use with a drive and still get unique data.	Head
High Level Language	A more sophisticated coding language than assembly language. One high level instruction may generate many machine instructions. (e.g. FORTRAN, BASIC, PASCAL, etc.)	High Level Language
Hostile Environment	A system software environment in which it is assumed that both hardware and software may fail in any way, and the system is required either to continue running or shut itself down in an orderly manner.	Hostile Environment

(APPENDIX G GLOSSARY continued)

Initialize	A disk is initialized such that PDOS parameters are available to the file manager. These include disk name, number of directory entries, total number of sectors available, date of initialization, density and sides flags, directory, and sector bit map. Any bad sectors are deallocated from user storage.	Initialize
Interleaving	A track formatting technique whereby multiple sectors may be read or written sequentially with a minimum of disk latency. This is possible by placing logical sectors on a track in such a way that the time required by the system service routine to process a single sector is less than the time required for the disk to rotate to the start of the next logical sector.	Interleaving
Interleave Factor	The number of physical sectors between a given sector and the next logical sector on a disk track.	Interleave Factor
Interpreter	A translation program used to carry out statements expressed in a high-level language. Usually its intermediate code cannot be directly executed on a general purpose processor.	Interpreter
Interrupt	A signal from an external source that causes the processor to stop execution of the current task, save current task status, and begin executing a system service routine or another user task.	Interrupt
Interrupt Mask	A processor defined variable which limits interrupt levels.	Interrupt Mask
Interval Timer	A hardware clock which generates an interrupt after a specified period of time has elapsed.	Interval Timer
I/O Channel	See File Slot.	I/O Channel

(APPENDIX G GLOSSARY continued)

Kernel	The most basic portion of an operating system, usually supporting only task scheduling, communication, coordination, and memory allocation.	Kernel
Linked List	A data structure in which each element contains a pointer to its predecessor or successor (singly linked) or both (doubly linked).	Linked List
Linker	A system software utility that connects previously assembled/compiled tasks or subroutines into a single object module that can be loaded into memory for execution.	Linker
Loader	A system software utility that moves object code from secondary storage into memory, performing relocation as required.	Loader
Logical Device	A reference to an I/O device by name or number without regard to the exact nature of the I/O device.	Logical Device
Logical Sector	A sector within a disk partition.	Logical Sector
Logical Track	A software address on the drive that appears to the operating system as good track data, which may or may not be the same as the physical track.	Logical Track
Mailbox	A system data structure that handles task communication through global memory buffers.	Mailbox
Memory Bit Map	PDOS uses a memory bit map for memory allocation and deallocation in 2k byte increments. See Bit Map.	Memory Bit Map
Memory Mapped	A method of implementing system I/O through memory locations.	Memory Mapped
Monitor	A monitor is a set of resident commands for handling the most common functions of the operating system.	Monitor

(APPENDIX G GLOSSARY continued)

Multi-tasking	The ability of an operating system to permit multiple tasks to run concurrently.	Multi-tasking
Multi-user	The ability of an operating system to multi-task and allow multiple users complete system access.	Multi-user
Non-preemptive Scheduling	A scheduling algorithm where a task does not stop executing until it is complete.	Non-preemptive Scheduling
Object Code	The output of an assembler or compiler that can be loaded and executed on the target processor.	Object Code
Open	A system service which allocates a file or resource to a task.	Open
Operating System	A collection of system software that permits user written tasks to interface to the machine hardware and interact with other tasks in a straightforward, efficient, and safe manner.	Operating System
Overhead	The amount of processing time required by the operating system to perform housekeeping such as paging, swapping, and scheduling. Or, the amount of memory required by the operating system to maintain tasks.	Overhead
Overlay	A technique used to execute programs which are larger than the available memory size in systems without paging or segmentation capabilities. In PDOS, FORTRAN permits overlays and PDOS BASIC can simulate overlays.	Overlay
Page	An indivisible segment of memory which facilitates memory management.	Page
Parameter List	A parameter list refers to parameters or variables used to pass information to a command.	Parameter List

(APPENDIX G GLOSSARY continued)

Partition	A logical division or portion of a mass storage device which can be addressed by PDOS using a disk number. Winchester drives are usually divided into some large and some small partitions, on track boundaries.	Partition
PDOS Sector	A logical sector to PDOS on a particular disk and ranges from 0 to 65535 ($FFFF). Only 65280 ($FF00) total sectors may be used for a PDOS file system.	PDOS Sector
Phantom Port	A user port that has no physical device associated with it (port 0).	Phantom Port
Physical Device	A physical device is a hardware unit such as a disk or tape drive. The operating system binds a physical device to a logical device. User routines reference logical devices rather than physical devices.	Physical Device
Physical Track	An actual combination of one head and one cylinder on the drive.	Physical Track
Position Independent Code	Executable code which runs independent of the physical memory location at which it is loaded.	Position Independent Code
Preemptive Scheduling	A scheduling technique where task scheduling is independent of task completion. Round-robin swapping or high priority tasks can interrupt task execution at any time.	Preemptive Scheduling
Program Counter	A register within the processing element of a computer that contains the address of the next instruction to be executed. It is automatically incremented by the processor and modified by transfer instructions.	Program Counter
Queue	A data structure in which the first element in is the first element out.	Queue

(APPENDIX G GLOSSARY continued)

Random Access	A type of file access in which data may be accessed in a random manner, regardless of its position within the file.	Random Access
Realtime	An action or system capable of action at a speed commensurate with the time of occurrence of an actual process. Events must be handled promptly (i.e., within set timing limits).	Realtime
Realtime Clock	A system clock that indicates actual elapsed time from some reference time.	Realtime Clock
Record	A set of data elements that are logically accessed together.	Record
Reentrant Code	Code that may be executed simultaneously by more than one task. The code cannot be modified during execution and each task must maintain its own data area.	Reentrant Code
Resource	Assets of a computer system that the operating system uses and/or allocates to tasks for their use. These include memory, disk storage, printers, and terminals, as well as processors.	Resource
Response Time	The elapsed time from the entry of a command until its acknowledgement or completion.	Response Time
Retry	An attempt to provide automatic error recovery by executing the failed operation a second time.	Retry
Roll in/ Roll out	Roll in / Roll out functions refer to moving buffers or tasks to and from secondary storage when limited resources are available.	Roll in/Roll out

(APPENDIX G GLOSSARY continued)

ROMable Code	Object code that is not self-modifying, will execute properly when placed in ROM, and which uses scratch pad RAM external to the code.	ROMable Code
Round-Robin Scheduling	A scheduling method where tasks in the task list are executed in order, and entries into the list are always put at the end. Each task is given a time limit for execution and executes the full time unless blocked or a swap call is made to the operating system.	Round-Robin Scheduling
Scheduler	A system service that determines which task within the system should be run next.	Scheduler
SCSI	Small Computer Systems Interface.	SCSI
Sector	The smallest contiguous storage area on a secondary storage medium. PDOS uses 256-byte logical sectors.	Sector
Sector Bit Map	PDOS uses a sector bit map on each secondary storage unit to allocate and deallocate logical sectors. See Bit Map.	Sector Bit Map
Sector Buffer	A buffer associated with a file slot for I/O transfers to and from secondary storage.	Sector Buffer
Semaphore	A "gating" variable that is used to synchronize task operations on shared data. (See critical code.)	Semaphore
Sequential File Access	A type of file access where data may only be read or written sequentially, one record at a time.	Sequential File Access
Soft Error	A dynamic error normally caused by some transient condition. Retrying the failed operation often results in successful completion.	Soft Error

(APPENDIX G GLOSSARY continued)

Source Code	Source code is ASCII text which is passed through a compiler or assembler to produce object code.	Source Code
Spawn	The spawn process generates a new task or entity. The new task is referred to as the spawned task.	Spawn
Static Priority	A task's execution priority is fixed either when the task is loaded or at time of system generation.	Static Priority
Status Register	A processor register containing the current executing conditions.	Status Register
Suspended	A task state in which task execution is discontinued pending the occurrence of an event.	Suspended
Swapping	The movement from one task to the next via the scheduler.	Swapping
Synchron-ization	The process of coordinating the execution of tasks within an operating system.	Synchronization
System Generation	The process of generating, linking, and loading all required system modules together in order to build a new operating system or to update tables in an existing system.	System Generation
System Service	Functions such as timekeeping, memory allocation, and console I/O that the operating system performs for user tasks upon request.	System Service
System Software	Software that is part of or closely associated with the operating system.	System Software
System Support	Functions or utilities such as language translators, debugging tools, diagnostics, and libraries which enable a system user or programmer to write and test tasks in an efficient manner.	System Support

(APPENDIX G GLOSSARY continued)

Target Machine	The final machine on which a program is run.	Target Machine
Task Control Block	A task control block (TCB) is a block of memory containing the information needed by the operating system to schedule, suspend, and support a task. This includes scratch pad areas, buffers, port assignments, and other information necessary for the operating system to be reentrant.	Task Control Block
Task List	A system data structure containing a list of tasks within the system. This information includes the minimal amount of data required to suspend and resume task execution.	Task List
Task Lock	The process of locking a task in the run state such that no other task executes until an unlock task is done.	Task Lock
Task State	The status of a task (i.e., ready, executing, suspended or undefined).	Task State
Throughput	The quantity of information processed by a computer system in a unit time.	Throughput
Time Slice	The smallest time quantity available to the operating system for use in task scheduling.	Time Slice
Trace	A trailing record of a program's execution.	Trace
Unit	A logical gating variable which directs characters to various output destinations.	Unit
Utility	A software program supplied with the operating system which supports program development.	Utility
Wait	A system service that causes a task to be suspended for a specified time or pending the occurrence of an event.	Wait
Wakeup	The act of making a task ready to run after a period of suspension.	Wakeup

VIRTUAL PORT INTERNALS

MSYRAM:SR

```
                                      ._____.
$0340 WIND. DS.L 1 ---------->| O.W  |
$0344 WADR. DS.L 1          #1|      |
                 \          #2|      |          ._________.
                  \         #3|      |      -->|         |
                   \         .....         /   |Screen #1|
                    \      #14|      |    /    | 24 x 80 |
                     \     #15|      |   /     |         |
                      \       '-----'   /      '---------'
                       \               /
                        \     ._____. /         ._________.
                          -->| O.L  |/    ---->|         |
                           #1|      |    /     |Screen #2|
                           #2|      |---/      | 24 x 80 |
                           #3|      |          |         |
                              .....            '---------'
                          #14|      |
                          #15|      |             .....
                              '-----'\
                              | O.L  | \        ._________.
                           #1| C/P  |  ------->|          |
                           #2|      |          |Screen #15|
                           #3|      |          | 24 x 80  |
                              .....            |          |
                          #14|      |          '----------'
                          #15|      |
                              '-----'
```

```
(WIND.).W = FRPM ____ D__p pppp
            \\\\ \\\\ \\\\    \_
             \\\\ \\\\ \\\\_____ 0-4=PORT #
              \\\\ \\\\ \\\_____ 5  = Reserved
               \\\\ \\\\ \\_____ 6  = Reserved
                \\\\ \\\\ \_____ 7  = WINDOWING DISABLE
                 \\\\ \\\\______ 8  = Reserved
                  \\\\ \\\______ 9  = Reserved
                   \\\\ \\______ 10 = Reserved
                    \\\\ \______ 11 = Reserved
                     \\\\_______ 12 = ALREADY DEFINED
                      \\\_______ 13 = PRINT FLAG
                       \\_______ 14 = REFRESH FLAG
                        \_______ 15 = LEAD FLAG
```

(H. VIRTUAL PORT INTERNALS continued)

PDOS Interrupt Input Processor

Normal:

```
                                Input Buffers
     Port #2                     ._______.
        |                        |  #1   |
        v                        |-------|
     K2$CHIN ------------------->|  #2   |
                                 |-------|
                                 |  #3   |
                                 |-------|
                                 |  #4   |
                                 |-------|
                                   .....
                                 |-------|
                                 |  #15  |
                                 '-------'
```

Windowing:

```
                (WIND.)
                 .___.
                 | - |          Input Buffers
     Port #2     |---|           ._______.
        |        |   |           |  #1   |
        v        |---|           |-------|
     K2$CHIN --> | 4 |--         |  #2   |
                 |---|   \       |-------|
                 |   |    \      |  #3   |
                 |---|     \     |-------|
                 |   |      --> |  #4   |
                 |---|           |-------|
                  ...              .....
                 |---|           |-------|
                 |   |           |  #15  |
                 '---'           '-------'
```

(H. VIRTUAL PORT INTERNALS continued)

PDOS Output Processor:

Normal:

```
             UxP$
Clear        .___.                                4
Position --->| 4 |-------------------------------> xBIOSU
Put char     '---'
```

Windowing:

```
                                    (WIND.)
                                      .___.
             UxP$                     |   |#1
Clear        .___.                    |---|    2
Position --->| 4 |--------------+---->| 4 |#2 ---> xBIOSU
Put char     '---'              ^     |---|
               |                |     |   |#3
               |                |     |---|
               |                |     |   |#4
               |                |     |---|
               |                |       ...
               |                |     |---|
               |                |     |   |#15
             4 |   (WADR.)      |     '---'
               |    .___.       |
               |    | - |       |
               |    |---|       | Event 127
               |    |#1 |       |     +
               |    |---|       |  Bit #13
               |    |#2 |       |
               |    |---|    .__|____.
               |    |#3 |    |       |
               |    |---|    |Window |
               '--->|#4 |--->|Image 4|
                    |---|    |       |
                    |#5 |    '-------'
                    |---|
                     ...
```

- A -

- B -

- H -

- I -

- K -

- L -

- T -