VMEPROM SYSTEM CALLS

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

1. VMEPROM SYSTEM CALLS

1.1 General Information

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
3) file support primitives.

The following primitives, which are available in a standard PDOS operating system environment are not available in VMEPROM:

XBUG	Calls the PDOS debugger, this module is not included in VMEPROM
XCHF	PDOS monitor command, not included in VMEPROM
XLST	PDOS monitor command, not included in VMEPROM
XBFL	PDOS monitor command, not included in VMEPROM
XAIM	PDOS monitor command, not included in VMEPROM
XGTP	PDOS monitor command, not included in VMEPROM
XEXZ	PDOS monitor command, not included in VMEPROM

These primitives give reference to the PDOS Monitor or PDOS Debugger and these modules are not included in VMEPROM.

The monitor calls XGNP and XPCB of PDOS are emulated by VMEPROM and perform their expected functions.

1.2 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.

Example for 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:

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
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
XEXC - Execute PDOS call D7.W
XLER - Load error register
XERR - Return error D0 to VMEPROM
XEXT - Exit to VMEPROM
XEXZ - Exit to VMEPROM with command

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

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.

XGCB - Conditional get character
XGCC - Get character conditional
XGCR - Get character
XGCP - Get port character
XGLB - Get line in buffer
XGLM - Get line in monitor buffer
XGLU - Get line in user 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.

XFFN - Fix file name
XLFN - Look for name in file slots
XBFL - Build file directory list
XRDE - Read next directory entry
XRDN - Read directory entry by name
XAPF - Append file
XCPY - Copy file
XLDF - Load file
XRCN - Reset console inputs
XRST - Reset disk
XSZF - Get disk size

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.

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.

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

1.3 Description of Kernel Primitives

This chapter gives a detailed description of all Kernel calls which are available in VMEPROM.

1.3.1 X881 - SAVE 68881 ENABLE

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

Description: 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 VMEPROM.

See also: None

Possible Errors: None

1.3.2 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.

Description: 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.

See also: None

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	

1.3.3 XBCP - BAUD CONSOLE PORT

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

Registers: In    D2.W = f0PI 8DBS / <port #>
                 D3.W = Baud rate
                 D1.W = Port type
                 D5.L = Port base
```

Description:

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: 1.3.78 XRPS - READ PORT STATUS
1.3.92 XSPF - SET PORT FLAG

Possible Errors:

66 = Invalid port or baud rate

1.3.4 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.

Description: 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.

See also: None

Possible Errors: None

1.3.5 XCBD - CONVERT BINARY TC DECIMAL

Mnemonic:	XCBD
Value:	$A050
Module:	MPDOSK3
Format:	XCBD

Registers:	In	D1.L = Number
	Out	(A1) = String

Description: 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: 1.3.9 XCBX - CONVERT TO DECIMAL IN BUFFER.

Possible Errors: None

1.3.6 XCBH - CONVERT BINARY TO HEX

Mnemonic:	XCBH
Value:	$A052
Module:	MPDOSK3
Format:	XCBH

```
Registers: In    D1.L = Number
           Out   (A1) = String
```

Description: 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: 1.3.12 XCHX - CONVERT BINARY TO HEX IN BUFFER.

Possible Errors: None

1.3.7 XCBM - CONVERT TO DECIMAL W/MESSAGE

Mnemonic:	XCBM
Value:	$A054
Module:	MPDOSK3
Format:	XCBM <message>

```
Registers: In   D1.L = Number
           Out  (A1) = String
```

Description: 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.

See also: None

Possible Errors: None

1.3.8 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.

Description: 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 VMEPROM.

An 'EQ' status indicates that no character was entered. An 'NE' status indicates a pause has occurred.

See also: None

Possible Errors: None

1.3.9 XCBX - CONVERT TO DECIMAL IN BUFFER

Mnemonic:	XCBX
Value:	$A06A
Module:	MPDOSK3
Format:	XCBX

Registers: In D1.L = Number
(A1) = Buffer

Description: 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: 1.3.5 XCBD - CONVERT BINARY TO DECIMAL.

Possible Errors: None

1.3.10 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.

Description:

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.

See also: None
Possible Errors: None

1.3.11 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

Description: 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.

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

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.

See also: 1.3.72 XRFA - READ FILE ATTRIBUTES
1.3.109 XWFA - WRITE FILE ATTRIBUTES
1.3.110 XWFP - WRITE FILE PARAMETERS

Possible Errors:

52 = File not open
59 = Invalid file slot
75 = File locked
Disk errors

1.3.12 XCHX - CONVERT BINARY TO HEX IN BUFFER

Mnemonic:	XCHX
Value:	$A068
Module:	MPDOSK3
Format:	XCHX

Registers: In D1.L = Number
(A1) = Output buffer

Description: 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: 1.3.6 XCBH - CONVERT BINARY TO HEX.

Possible Errors: None

1.3.13 XCLF - CLOSE FILE

Mnemonic:	XCLF
Value:	$A0D2
Module:	MPDOSF
Format:	XCLF <status error return>

Registers: In D1.W = File ID

Description:

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.

See also: None

Possible Errors:

52 = File not open
59 = Invalid slot #
75 = File locked
Disk errors

1.3.14 XCLS - CLEAR SCREEN

Mnemonic:	XCLS
Value:	$A076
Module:	MPDOSK2
Format:	XCLS

Registers: None

Note: The clear screen characters are located in the user TCB variable CSC$(A6).

Description: 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.

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 ST command maintains the CSC$ field, although it can be altered under program control.

See also: 1.3.67 XRCP - READ PORT CURSOR POSITION

Possible Errors: None

1.3.15 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.

Description: 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.

See also: None

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

1.3.16 XCTB - CREATE TASK BLOCK

```
Mnemonic:     XCTB
Value:        $A026
Module:       MPDOSK1
Format:       XCTB
              <status error return>

Registers: In    D0.W = Task size (1 Kbyte 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.

Description:

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 1 Kbytes) 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 1 Kbytes. Address register A2 points to the new task command line. If A2 is zero, then VMEPROM 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 $1000 bytes at the low end and enough user stack space at the upper end.

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 VMEPROM is invoked). 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 VMEPROM 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.

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.

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.

See also: None

Possible Errors:

72 = Too many tasks
73 = Not enough memory

1.3.17 XDEV - DELAY SET/RESET EVENT

```
Mnemonic:     XDEV
Value:        $A032
Module:       MPDOSK1
Format:       XDEV
              <status error return>

Registers: In   D0.L = Time
                D1.B = Event (+=Set, -=Reset)
```

Note: If D0.L=0, then the D1.B event is cleared.

Description:

The DELAY SET/RESET EVENT primitive places a timed event in a system stack controlled by the system clock. Data register D0.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 D0 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:

1.3.88 XSEF - SET EVENT FLAG W/SWAP
1.3.89 XSEV - SET EVENT FLAG
1.3.95 XSUI - SUSPEND UNTIL INTERRUPT
1.3.100 XTEF - TEST EVENT FLAG

Possible Errors:

83 = Delay event stack full

1.3.18 XDFL - DEFINE FILE

Mnemonic: XDFL
Value: $A0D4
Module: MPDOSF
Format: XDFL
<status error return>

Registers: In D0.W = # of contiguous sectors
(A1) = File name

Description: 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.

See also: None

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

1.3.19 XDLF - DELETE FILE

Mnemonic: XDLF
Value: $A0D6
Module: MPDOSF
Format: XDLF
<status error return>

Registers: In (A1) = File name

Description: 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.

See also: None

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

1.3.20 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
```

Description:

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: None

Possible Errors: None

1.3.21 XDPE - DELAY PHYSICAL EVENT

Mnemonic:	XDPE
Value:	$A114
Module:	MPDOSK1
Format:	XDPE

Registers: In A0 = Event address
D0.L = Time TICs for delay (0=clear entry)
D1.W = Event descriptor

Description: XDPE causes the specified event to be set/cleared after the specified time has elapsed. Each event can have only one delayed action pending. Successive calls will supersede pending requests. Only the lower eight bits of the descriptor are used. To cancel pending actions, specify a delay time of 0.

The event descriptor is a 16-bit word that defines both the bit number at the specified A0 address and the action to take on the bit. The following bits are defined:

```
Bit number -- 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
               T  x  x  x  x  x x x S x x x x B B B

               T = Should the bit be toggled on scheduling?
               1 = Yes (toggle),    0 = No (do not toggle)

               S = Suspend on event bit clear or set
               1 = Suspend on SET, 0 = Suspend on CLEAR

             BBB = The 680 x 0 bit number to use as an event
               x = Reserved, should be 0
```

Since the bit number is specified in the lower three bits of the descriptor, you may use the descriptor with the 680 x 0 BTST, BCLR, BSET instructions.

See also: XDEV - Delay Set/Clear Event
XSOE - Suspend on Physical Event
XTLP - Translate Logical to Physical Event

1.3.22 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.

```
D1.L = TVCZ FEDCBA9876543210
         \\\\ \             \_
          \\\\ \_______________ TRAPs #0-#15
           \\\\________________ Zero divide
            \\\________________ CHK
             \\________________ TRAPV
              \________________ Trace exception
```

Description:

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.

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).

See also:

Possible Errors: None

1.3.23 XERR - RETURN ERROR D0 TO VMEPROM

Mnemonic: XERR
Value: $A00C
Module: MPDOSK1
Format: XERR

Registers: In D0.W = Error code

Description: The RETURN ERROR D0 TO VMEPROM primitive exits to VMEPROM 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:
1.3.24 XEXT - EXIT TO VMEPROM

Possible Errors: None

1.3.24 XEXC - EXECUTE PDOS CALL D7.W

Mnemonic:	XEXC
Value:	$A030
Module:	MPDOSK1
Format:	XEXC

Registers: In D7.W = Aline PDOS CALL

Description: 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.

See also:

Possible Errors: Call dependent

1.3.25 XEXT - EXIT TO VMEPROM

Mnemonic:	XEXT
Value:	$A00E
Module:	MPDOSK1
Format:	XEXT (Always exits to VMEPROM)

Registers: None

Description: The EXIT TO VMEPROM primitive exits a user program and returns to VMEPROM.

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:
1.3.22 XERR - RETURN ERROR D0 TO VMEPROM

Possible Errors: None

1.3.26 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
```

Description:

The FILE ALTERED CHECK primitive looks at the altered 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.

See also: None

Possible Errors: Disk errors

1.3.27 XFBF - FLUSH BUFFERS

Mnemonic:	XFBF
Value:	$A0F8
Module:	MPDOSF
Format:	XFBF <status error return>

Registers: None

Description: 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.

See also: None

Possible Errors: Disk errors

1.3.28 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
```

Description:

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

System defaults are used for the disk number and file directory level when they are not specified in the file name.

See also: 1.3.70 XRDN - READ DIRECTORY ENTRY BY NAME

Possible Errors:

50 = Invalid file name

1.3.29 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
```

Description: 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:
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE
1.3.106 XUTM - UNPACK TIME

Possible Errors: None

1.3.30 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

Description: 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

1.3.31 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].

Description: 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

1.3.32 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].

Description: 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

1.3.33 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].

Description:

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

1.3.34 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].

Description: 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

1.3.35 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].

Description: 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 VMEPROM 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: 1.3.36 XGLU - GET LINE IN USER BUFFER
Possible Errors: None

1.3.36 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].

Description:

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 VMEPROM 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

1.3.37 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].

Description:

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 VMEPROM 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

1.3.38 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

Description: 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

1.3.39 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
```

Description: 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:

1.3.40 XGTM - GET TASK MESSAGE
1.3.44 XKTM - KILL TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message slot empty

1.3.40 XGNP - GET NEXT PARAMETER

```
Mnemonic:     XGNP
Value:        $A05A
Module:       Emulated by VMEPROM
Format:       XGNP
              <status return>

Registers: Out  SR = LO....No parameter
                           [(A1)=0]
                     EQ....Null Parameter
                           [(A1)=0]
                     HI....Parameter
                           [(A1)=PARAMETER]
```

Description:

The GET NEXT PARAMETER primitive parses the VMEPROM command buffer for the next command parameter. 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.

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.

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.

A 'HI' status is returned if a valid parameter is found. Address register A1 then points to the parameter.

Possible Errors: None

1.3.41 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
```

Description: 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:

1.3.38 XGMP - GET MESSAGE POINTER
1.3.44 XKTM - KILL TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors: None

1.3.42 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
```

Description: The GET USER MEMORY primitive searches the system memory bit map for a contiguous block of memory equal to D0.W Kbytes. If found, the 'EQ' status is set, address registers A0 and A1 are returned the start and end memory address, and the memory block is marked as allocated in the bit map.

See also: 1.3.29 XFUM - FREE USER MEMORY

Possible Errors:

73 = Not enough memory

1.3.43 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

Description: 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:
1.3.79 XRSE - READ SECTOR
1.3.82 XRSZ - READ SECTOR ZERO
1.3.112 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

1.3.44 XKTB - KILL TASK

Mnemonic:	XKTB
Value:	$A0FA
Module:	MPDOSK1
Format:	XKTB <status error return>

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.

Description: 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.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.

See also: 1.3.16 XCTB - CREATE TASK BLOCK

Possible Errors:

74	= No such task
76	= Task locked

1.3.45 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
```

Description: The KILL TASK MESSAGE primitive allows you to read (and thus clear) any task's messages from the system message buffers.

See also:

1.3.38 XGMP - GET MESSAGE POINTER
1.3.40 XGTM - GET TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors: None

1.3.46 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.
```

Description:

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	

1.3.47 XLER - LOAD ERROR REGISTER

Mnemonic:	XLER
Value:	$A03A
Module:	MPDOSK1
Format:	XLER

Registers: In D0.W = Error number

Description: 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

1.3.48 XLFN - LOOK FOR NAME IN FILE SLOTS

Mnemonic:	XLFN
Value:	$A0A2
Module:	MPDOSF
Format:	XLFN <status return>

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.

Description: 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

1.3.49 XLKF - LOCK FILE

Mnemonic:	XLKF
Value:	$A0D8
Module:	MPDOSF
Format:	XLKF <status error return>

Registers: In   D1.W = File ID

Description:

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: 1.3.103 XULF - UNLOCK FILE

Possible Errors:

52	=	File not open
59	=	Invalid slot #
75	=	File locked

Disk errors

1.3.50 XLKT - LOCK TASK

Mnemonic:	XLKT
Value:	$A014
Module:	MPDOSK1
Format:	XLKT <status return>

```
Registers: Out  SR = EQ...Not locked
                     NE...Locked
```

Description: 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: 1.3.104 XULT - UNLOCK TASK

Possible Errors: None

1.3.51 XLSR - LOAD STATUS REGISTER

Mnemonic:	XLSR
Value:	$A02E
Module:	MPDOSK1
Format:	XLSR

Registers: In D1.W = 68000 status register

Description: 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: 1.3.96 XSUP - ENTER SUPERVISOR MODE

Possible Errors: None

1.3.52 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.

Description:

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.

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

1.3.53 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
```

Description: 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:

1.3.28 XFTD - FIX TIME & DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE

Possible Errors: Status errors.

1.3.54 XPBC - PUT BUFFER TO CONSOLE

Mnemonic:	XPBC
Value:	$A084
Module:	MPDOSK2
Format:	XPBC

Registers: None

Description: 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: 1.3.34 XGLB - GET LINE IN BUFFER

Possible Errors: None

1.3.55 XPCC - PUT CHARACTER(S) TO CONSOLE

Mnemonic:	XPCC
Value:	$A086
Module:	MPDOSK2
Format:	XPCC

Registers: In D0.W = Character(s)

Description:

The PUT CHARACTER TO CONSOLE primitive outputs one or two ASCII characters in data register D0 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:
1.3.56 XPCR - PUT CHARACTER RAW
1.3.57 XPDC - PUT DATA TO CONSOLE

Possible Errors: None

1.3.56 XPCL - PUT CRLF TO CONSOLE

Mnemonic:	XPCL
Value:	$A088
Module:	MPDOSK2
Format:	XPCL
Registers:	None

Description:

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

1.3.57 XPCP - PLACE CHARACTER IN PORT BUFFER

```
Mnemonic:     XPCP
Value:        $AOBC
Module:       MPDOSK2
Format:       XPCP

Registers: In    D0.B  = Character to insert
                 D1.W  = Input port number (1 to 15)
           Out   SR      = .EQ.   =  High water  (character is
inserted)
                         .NE.  = Character is inserted
```

Description: XPCP allows a character to be placed into the input buffer of any VMEPROM port from a task or program.

Note: Once the status returns EQ (high water)_, subsequent XPCP calls will return a status of NE as if everything were normal, but the data is discarded. Once the status of EQ is detected, the transmitting task should monitor the status of the port with the XRPS (read port status) call until bit 56 is cleared.

The port specified in the XPCP call is independent of window g - it refers to the physical port, not the logical port.

1.3.58 XPCR - PUT CHARACTER RAW

Mnemonic: XPCR
Value: $A0BA
Module: MPDOSK2
Format: XPCR

Registers: In D0.B = CHARACTER

Description: The PUT CHARACTER RAW primitive outputs the character in the lower byte of data register D0 to the user console. No attempt is made by PDOS to interpret control characters.

See also:

1.3.54 XPCC - PUT CHARACTER(S) TO CONSOLE
1.3.57 XPDC - PUT DATA TO CONSOLE

Possible Errors: None

1.3.59 XPDC - PUT DATA TO CONSOLE

Mnemonic:	XPDC
Value:	$A096
Module:	MPDOSK2
Format:	XPDC

Registers: In D7.W = LENGTH
(A1) = DATA STRING

Description: 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:
1.3.54 XPCC - PUT CHARACTER(S) TO CONSOLE
1.3.56 XPCR - PUT CHARACTER RAW

Possible Errors: None

1.3.60 XPEL - PUT ENCODED LINE TO CONSOLE

Mnemonic:	XPEL
Value:	$A06E
Module:	MPDOSK2
Format:	XPEL

Registers: In (A1) = Message

Description:

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:

1.3.59 XPEM - PUT ENCODED MESSAGE TO CONSOLE
1.3.60 XPLC - PUT LINE TO CONSOLE
1.3.61 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

1.3.61 XPEM - PUT ENCODED MESSAGE TO CONSOLE

Mnemonic:	XPEM	
Value:	$A09C	
Module:	MPDOSK2	
Format:	XPEM	<message>

Registers: None

Description:

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:

1.3.58 XPEL - PUT ENCODED LINE TO CONSOLE
1.3.60 XPLC - PUT LINE TO CONSOLE
1.3.61 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

1.3.62 XPLC - PUT LINE TO CONSOLE

Mnemonic:	XPLC
Value:	$A08A
Module:	MPDOSK2
Format:	XPLC

Registers: In (A1) = ASCII string

Description: 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:

1.3.58 XPEL - PUT ENCODED LINE TO CONSOLE
1.3.59 XPEM - PUT ENCODED MESSAGE TO CONSOLE
1.3.61 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

1.3.63 XPMC - PUT MESSAGE TO CONSOLE

Mnemonic:	XPMC	
Value:	$A08C	
Module:	MPDOSK2	
Format:	XPMC	<message>

Registers: None

Description:

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:

1.3.58 XPEL - PUT ENCODED LINE TO CONSOLE
1.3.59 XPEM - PUT ENCODED MESSAGE TO CONSOLE
1.3.60 XPLC - PUT LINE TO CONSOLE

Possible Errors: None

1.3.64 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.

Description: 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.

The rcw 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.

The ST command of the user interface can be used to change the position cursor codes.

See also:
1.3.14 XCLS - CLEAR SCREEN
1.3.67 XRCP - READ PORT CURSOR POSITION

Possible Errors: None

1.3.65 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.

Description:

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:

1.3.73 XRFP - READ FILE POSITION
1.3.87 XRWF - REWIND FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
70 = Position error
Disk errors

1.3.66 XPSP - PUT SPACE TO CONSOLE

Mnemonic:	XPSP
Value:	$A098
Module:	MPDOSK2
Format:	XPSP

Registers: None

Description: 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: 1.3.54 XPCC - PUT CHARACTER(S) TO CONSOLE

Possible Errors: None

1.3.67 XRBF - READ BYTES FROM FILE

Mnemonic: XRBF
Value: $A0DE
Module: MPDOSF
Format: XRBF
<status error return>

```
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.)
```

Description: 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:

1.3.74 XRLF - READ LINE FROM FILE
1.3.107 XWBF - WRITE BYTES TO FILE
1.3.111 XWLF - WRITE LINE TO FILE

Possible Errors:

52 = File not open
56 = End of file
59 = Invalid slot #
Disk errors

1.3.68 XRCN - RESET CONSOLE INPUTS

Mnemonic:	XRCN
Value:	$A0B2
Module:	MPDOSF
Format:	XRCN

Registers: None

Description: The RESET CONSOLE INPUTS closes the current procedure file. If there are other procedure files pending (nested), then they become active again.

See also: 1.3.4 XCBC - CHECK FOR BREAK CHARACTER

Possible Errors: None

1.3.69 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.

Description: 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:
1.3.14 XCLS - CLEAR SCREEN
1.3.62 XPSC - POSITION CURSOR

Possible Errors: None

1.3.70 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
```

Description:

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

1.3.71 XRDM - DUMP REGISTERS

Mnemonic:	XRDM
Value:	$A02A
Module:	MPDOSK1
Format:	XRDM

Registers: In All

Description: 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: 1.3.20 XDMP - DUMP MEMORY FROM STACK

Possible Errors: None

1.3.72 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
```

Description: 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: 1.3.27 XFFN - FIX FILE NAME

Possible Errors:

53 = File not defined
68 = Not PDOS disk
Disk errors

1.3.73 XRDT - READ DATE

Mnemonic:	XRDT
Value:	$A05C
Module:	MPDOSK3
Format:	XRDT

Registers: Out (A1) = 'MN/DY/YR'<null>

Description: 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:

1.3.28 XFTD - FIX TIME & DATE
1.3.52 XPAD - PACK ASCII DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE
1.3.106 XUTM - UNPACK TIME

Possible Errors: None

1.3.74 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.

Description: 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:

1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.109 XWFA - WRITE FILE ATTRIBUTES
1.3.110 XWFP - WRITE FILE PARAMETERS

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
Disk errors

1.3.75 XRFP - READ FILE POSITION

Mnemonic:	XRFP
Value:	$A0FE
Module:	MPDOSF
Format:	XRFP <status error return>

```
Registers: In    D1.W = File ID
           Out   (A3) = File slot address
                 D2.L = Byte position
                 D3.L = EOF byte position
```

Description: 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:

1.3.63 XPSF - POSITION FILE
1.3.87 XRWF - REWIND FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
Disk errors

1.3.76 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.)
```

Description:

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:

1.3.65 XRBF - READ BYTES FROM FILE
1.3.107 XWBF - WRITE BYTES TO FILE
1.3.111 XWLF - WRITE LINE TO FILE

Possible Errors:

52 = File not open
56 = End of file
59 = Invalid slot #
Disk errors

1.3.77 XRNF - RENAME FILE

Mnemonic: XRNF
Value: $A0E4
Module: MPDOSF
Format: XRNF

<status error return>

Registers: In (A1) = Old file name
(A2) = New file name

Description: 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:

1.3.18 XDFL - DEFINE FILE
1.3.19 XDLF - DELETE FILE

Possible Errors:

50 = Invalid file name
51 = File already defined
Disk errors

1.3.78 XROO - OPEN RANDOM READ ONLY FILE

Mnemonic:	XROO
Value:	$A0E6
Module:	MPDOSF
Format:	XROO <status error return>

```
Reqisters: In    (A1) = File name
           Out  D0.W = File attribute
                D1.W = File ID
```

Note: Uses multiple directory file search.

Description: 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.

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
```

1.3.79 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.

Description:

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.

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

1.3.80 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.

Description: 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$.

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:

1.3.3 XBCP - BAUD CONSOLE PORT
1.3.92 XSPF - SET PORT FLAG

Possible Errors:

66 = Invalid port or baud rate

1.3.81 XRSE - READ SECTOR

Mnemonic:	XRSE
Value:	$A0C2
Module:	MPDOSF
Format:	XRSE <status error return>

Registers: In D0.B = Disk number
D1.W = Sector number
(A2) = Buffer pointer

Description: 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:
1.3.42 XISE - INITIALIZE SECTOR
1.3.82 XRSZ - READ SECTOR ZERO
1.3.112 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

1.3.82 XRSR - READ STATUS REGISTER

Mnemonic:	XRSR
Value:	$A042
Module:	MPDOSK1
Format:	XRSR

Registers: Out D0.W = 68000 status register

Description: 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

1.3.83 XRST - RESET DISK

Mnemonic:	XRST
Value:	$A0B4
Module:	MPDOSF
Format:	XRST

```
Registers: In   D1.W = -1.... Reset by task
                       >=0... Reset by disk
```

Description: 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:
1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.13 XCLF - CLOSE FILE

Possible Errors: None

1.3.84 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)
```

Description: 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:

1.3.42 XISE - INITIALIZE SECTOR
1.3.79 XRSE - READ SECTOR
1.3.112 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

1.3.85 XRTE - RETURN FROM INTERRUPT

Mnemonic:	XRTE
Value:	$A044
Module:	MPDOSK1
Format:	XRTE

Registers: In   SSP = Status register.W
                      Program counter.L

Description: 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

1.3.86 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.)
```

Description: 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:

1.3.28 XFTD - FIX TIME & DATE
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE
1.3.106 XUTM - UNPACK TIME

Possible Errors: None

1.3.87 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
```

Description: 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:
1.3.28 XFTD - FIX TIME & DATE
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE
1.3.101 XUTM - UNPACK TIME

Possible Errors: None

1.3.88 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.

Description: 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: 1.3.94 XSTP - SET/READ TASK PRIORITY

Possible Errors: None

1.3.89 XRWF - REWIND FILE

Mnemonic: XRWF
Value: $A0EA
Module: MPDOSF
Format: XRWF
<status error return>

Registers: In D1.W = File ID

Description: The REWIND FILE primitive positions the file specified by the file ID in register D1, to byte position zero.

See also:
1.3.63 XPSF - POSITION FILE
1.3.73 XRFP - READ FILE POSITION

Possible Errors:

52 = File not open
59 = Invalid slot #
70 = Position error
Disk errors

1.3.90 <u>XSEF - SET EVENT FLAG W/SWAP</u>

Mnemonic: XSEF
Value: $A018
Module: MPDOSK1
Format: XSEF
<status return>

Registers: In D1.B = Event (+=Set, -=Reset)
Out SR = NE....Set
EQ....Reset

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.

Description: 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 module 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.)

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

See also:
1.3.17 XDEV - DELAY SET/RESET EVENT
1.3.89 XSEV - SET EVENT FLAG
1.3.95 XSUI - SUSPEND UNTIL INTERRUPT
1.3.100 XTEF - TEST EVENT FLAG

Possible Errors: None

1.3.91 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

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.
```

Description: The SET EVENT FLAG primitive sets or resets an event flag bit. The event number is specified in data register D1.B and is module 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.)

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

See also:

```
        1.3.17 XDEV  - DELAY SET/RESET EVENT
        1.3.89 XSEV  - SET EVENT FLAG
        1.3.95 XSUI  - SUSPEND UNTIL INTERRUPT
        1.3.100 XTEF - TEST EVENT FLAG
```

Possible Errors: None

1.3.92 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
```

Description:

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:

1.3.38 XGMP - GET MESSAGE POINTER
1.3.40 XGTM - GET TASK MESSAGE
1.3.44 XKTM - KILL TASK MESSAGE
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message buffer pending

1.3.93 XSOE - SUSPEND ON PHYSICAL EVENT

Mnemonic:	XSOE
Value:	$A112
Module:	MPDOSK1
Format:	XSOE

Registers: In	D1.L	=	Event 1 Descriptor.w, Event 0 Descriptor.w
	A0	=	Event 0 address (0=no event 0 to suspend on)
	A1	=	Event 1 address (0=no event 1 to suspend on)
Out	D0	=	-1 if awaken on event 0;1 if awaken on event 1

Note: This call is the same as XSUI but with physical events.

Description: XSOE allows a task to suspend on one or two events within the system. Tasks that suspend on physical events are listed as suspended on events -1/1. If event 0 is the scheduling event, a -1 is returned; otherwise, a 1 is returned.

The event descriptor is a 16 bit word that defines both the bit number at the specified A0,A1 address and the action to take o n the bit. The following bits are defined:

Bit number --	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	T	x	x	x	x	x	x	x	S	x	x	x	x	B	B	B

T = Should the bit be toggled on scheduling?
1 = Yes (toggle), 0 = No (do not toggle)

S = Suspend on event bit clear or set
1 = Suspend on SET, 0 = Suspend on CLEAR

BBB = The 680 x 0 bit number to use as an event
x = Reserved, should be 0

Since the bit number is specified in the lower three bits of the descriptor, you may use the descriptor with the 680x0 BTST,BCLR,BSET instructions.

See also: XDPE - Delay On Physical Event
XTLP - Translate Logical To Physical Event

1.3.94 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.

Description: 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.

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
```

1.3.95 XSPF - SET PORT FLAG

Mnemonic:	XSPF
Value:	$A09A
Module:	MPDOSK2
Format:	XSPF <status error return>

```
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.

Description: 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:

1.3.3 XBCP - BAUD CONSOLE PORT
1.3.78 XRPS - READ PORT STATUS

Possible Errors:

66 = Invalid port or baud rate

1.3.96 XSTM - SEND TASK MESSAGE

```
Mnemonic:     XSTM
Value:        $A020
Module:       MPDOSK1
Format:       XSTM
              <status error return>

Registers: In    D0.B = TASK NUMBER
                 (A1) = MESSAGE
```

Description:

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:

1.3.38 XGMP - GET MESSAGE POINTER
1.3.40 XGTM - GET TASK MESSAGE
1.3.44 XKTM - KILL TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors:

78 = Message buffer full

1.3.97 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.

Description: 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: 1.3.86 XRTS - READ TASK STATUS

Possible Errors:

74 = No such task

1.3.98 XSUI - SUSPEND UNTIL INTERRUPT

Mnemonic:	XSUI
Value:	$A01C
Module:	MPDOSK1
Format:	XSUI

Registers: In D1.W = EV1/EV2
Out D0.L = Event

Description:

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:
1.3.17 XDEV - DELAY SET/RESET EVENT
1.3.88 XSEF - SET EVENT FLAG W/SWAP
1.3.89 XSEV - SET EVENT FLAG
1.3.100 XTEF - TEST EVENT FLAG

Possible Errors: None

1.3.99 XSUP - ENTER SUPERVISOR MODE

Mnemonic:	XSUP
Value:	$A02C
Module:	MPDOSK1
Format:	XSUP

Registers: None

Description: 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:

1.3.50 XLSR - LOAD STATUS REGISTER
1.3.105 XUSP - RETURN TO USER MODE

Possible Errors: None

1.3.100 XSWP - SWAP TO NEXT TASK

Mnemonic:	XSWP
Value:	$A000
Module:	MPDOSK1
Format:	XSWP

Registers: None

Description: 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

1.3.101 XSZF - GET DISK SIZE

```
Mnemonic:     XSZF
Value:        $A0B6
Module:       MPDOSF
Format:       XSZF
              <status error return>

Registers: In   D0.B = Disk number
           Out  D5.L = Directory size/# of files
                D6.L = Allotted/Used
                D7.L = Largest/Free
```

Description:

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

1.3.102 XTAB - TAB TO COLUMN

Mnemonic:	XTAB	
Value:	$A090	
Module:	MPDOSK2	
Format:	XTAB	<column>

Registers: None

Description: 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

1.3.103 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)
```

Description: 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 module 128. Event 128 is local to each task.

See also:

1.3.17 XDEV - DELAY SET/RESET EVENT
1.3.88 XSEF - SET EVENT FLAG W/SWAP
1.3.89 XSEV - SET EVENT FLAG
1.3.95 XSUI - SUSPEND UNTIL INTERRUPT

Possible Errors: None

1.3.104 XTLP - TRANSLATE LOGICAL TO PHYSICAL EVENT

Mnemonic:	XTLP
Value:	$A110
Module:	MPDOSK1
Format:	XTLP

Registers:	In	D1.W	= Event 1.B,,Event 0.B
	Out	A0	= Event 0 address (0=no event 0 to suspend on)
		A1	= Event 1 address (0=no event 1 to suspend on)
		D1	= Event 1 Descriptor.w,Event 0 Descriptor.w

Description:

XTLP takes a VMEPROM logical event number and translates the event into a physical event. This call is used when a program needs to suspend on both a logical and a physical event. The logical event is first translated; then the XSOE call is used to suspend it.

A VMEPROM logical event is one of the 128 events maintained by the VMEPROM system in SYRAM.

Events are summarized as follows:

1 - 63	= Software events
64 - 80	= Software self clearing events
81 - 95	= Output port events
96 -111	= Input port events
112 -115	= Timer events
116 -127	= System control events
128	= Local

The event descriptor is a 16-bit word that defines both the bit number at the specified A0,A1 address and the action to take on the bit. The following bits are defined:

```
Bit number -- 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
              T  x  x  x  x  x  x x S x x x x B B B
```

T = Should the bit be toggled on scheduling?
1 = Yes (toggle), 0 = No (do not toggle)

S = Suspend on event bit clear or set
1 = Suspend on SET, 0 = Suspend on CLEAR

BBB = The 680 x 0 bit number to use as an event
x = Reserved, should be 0

Since the bit number is specified in the lower three bits of the descriptor, you may use the descriptor with the 680 x 0 BTST, BCLR, BSET instructions. You may also use the following physical manipulation calls which are macros for single assembly instructions. They are optimal as long as the values have already been placed in the correct registers. Physical events may need synchronization via the XTAS macro to avoid corruption. The macros are defined in the file PESMACS:SR.

XTST - Test Physical Event (replaces BTST D1, A0))
XSET - Test and Set Physical Event (replaces BSET D1,(A0))
XCLR - Test and Clear Physical Event (replaces BCLR D1,(A0))

Input: D1.W - Event descriptor
A0 - Event address
Output: None
Status: EQ - the bit was clear (0)
NE - the bit was set (1)

The bottom three bits are evaluated as a bit number. The bit at the address is set and the previous value is returned in the Z bit of the status register.

XTAS - Test and Set Physical Event (Bit 7 atomic)

This macro replaces TAS (A0). The seventh bit at the address is set and the previous value is returned in the N bit of the status register.

Input: A0 - Event address
Output: None
Status: EQ - the bit was clear (0)
NE - the bit was set (1)

See also: XDPE - Delay On Physical Event
XSOE - Suspend On Physical Event

1.3.105 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)
```

Description: 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:

```
        1.3.28  XFTD - FIX TIME & DATE
        1.3.52  XPAD - PACK ASCII DATE
        1.3.71  XRDT - READ DATE
        1.3.84  XRTM - READ TIME
        1.3.102 XUDT - UNPACK DATE
        1.3.106 XUTM - UNPACK TIME
```

Possible Errors: None

1.3.106 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>
```

Description: 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:
1.3.28 XFTD - FIX TIME & DATE
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.106 XUTM - UNPACK TIME

Possible Errors: None

1.3.107 XULF - UNLOCK FILE

Mnemonic:	XULF
Value:	$A0EE
Module:	MPDOSF
Format:	XULF <status error return>

Registers: In   D1.W = File ID

Description: 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: 1.3.48 XLKF - LOCK FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
Disk errors

1.3.108 XULT - UNLOCK TASK

Mnemonic: XULT
Value: $A016
Module: MPDOSK1
Format: XULT

Registers: None

Description: 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:
1.3.49 XLKT - LOCK TASK

Possible Errors: None

1.3.109 XUSP - RETURN TO USER MODE

Mnemonic:	XUSP
Value:	$A008
Module:	MPDOSK1
Format:	XUSP

Registers: None

Description: 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:

1.3.50 XLSR - LOAD STATUS REGISTER
1.3.96 XSUP - ENTER SUPERVISOR MODE

Possible errors: None

1.3.110 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>
```

Description: 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:

1.3.28 XFTD - FIX TIME & DATE
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE

Possible Errors: None

1.3.111 XVEC - SET/READ EXCEPTION VECTOR

Mnemonic:	XVEC
Value:	$A116
Module:	MPDOSK1
Format:	XVEC

```
Registers: In   D0.W  = Exception number (#2-255)
                (A0)  = New exception service routine (0=read

                        only)
           Out  (A0)  = Old service routine
```

Description: XVEC sets and/or reads the execution vector for the system. The old service routine address is returned so that you may change a routine and then restore the former routine under program control.

See also: XDTV - Define Trap Vectors

Possible Errors: None

1.3.112 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
```

Description:

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.

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.

See also:

1.3.65 XRBF - READ BYTES FROM FILE
1.3.74 XRLF - READ LINE FROM FILE
1.3.111 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

1.3.113 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)

Description: 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

1.3.114 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.
```

Description: 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:

1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.72 XRFA - READ FILE ATTRIBUTES
1.3.110 XWFP - WRITE FILE PARAMETERS

Possible Errors:

50 = Invalid file name
53 = File not defined
54 = Invalid file type
Disk errors

1.3.115 XWFP - WRITE FILE PARAMETERS

Mnemonic: XWFP
Value: $A0FC
Module: MPDOSF
Format: XWFP
<status error return>

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)

Description: 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:
1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.72 XRFA - READ FILE ATTRIBUTES
1.3.109 XWFA - WRITE FILE ATTRIBUTES

Possible Errors:

50 = Invalid file name
53 = File not defined
Disk errors

1.3.116 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

Description:

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.

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.

See also: 1.3.65 XRBF - READ BYTES FROM FILE
1.3.74 XRLF - READ LINE FROM FILE
1.3.107 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

1.3.117 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

Description: 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
1.3.42 XISE - INITIALIZE SECTOR
1.3.79 XRSE - READ SECTOR
1.3.82 XRSZ - READ SECTOR ZERO

Possible Errors:

Disk errors

1.3.118 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)

Description: 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

1.3.119 XZFL - ZERO FILE

```
Mnemonic:     XZFL
Value:        $A0F6
Module:       MPDOSF
Format:       XZFL
              <status error return>

Registers: In   (A1) = File name
```

Description: 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:
1.3.18 XDFL - DEFINE FILE
1.3.19 XDLF - DELETE FILE

Possible errors:

50 = Invalid file name
61 = File already open
68 = Not PDOS disk
Disk errors