PDOS ASSEMBLY
PRIMITIVES
REFERENCE
# Table of Contents

## Introduction

Guidelines For 68000 Assembly Programming .................................................. 1

PDOS Assembly Language Calls ................................................................. 4
  - System Calls ................................................................. 5
  - Console I/O Calls ............................................................ 6
  - System support calls ......................................................... 6
  - File Support Calls ........................................................... 7
  - File Management Calls ....................................................... 7
  - Disk Access Calls ............................................................. 8

PDOS Errors ..................................................................................... 9

## PDOS Assembly Primitives Reference

- X881 ................................................................................ 11
- XAPF ................................................................................ 12
- XBCP ................................................................................ 13
- XBFL ................................................................................ 15
- XBUG ................................................................................ 17
- XCBC ................................................................................ 19
- XCBHD ............................................................................... 20
- XCBH ............................................................................... 21
- XCBM ............................................................................... 22
- XCBP ............................................................................... 23
- XCBX ............................................................................... 24
- XCDB ............................................................................... 25
- XCDB ............................................................................... 26
- XCFD ............................................................................... 28
- XCHF ............................................................................... 29
- XCHX ............................................................................... 30
- XCLF ............................................................................... 31
- XCLS ............................................................................... 32
- XCPY ............................................................................... 33
- XCTB ............................................................................... 34
- XDEV ............................................................................... 37
- XDFL ............................................................................... 38
- XDLF ............................................................................... 40
- XDMP ............................................................................... 41
- XDPE ............................................................................... 42
- XDTH ............................................................................... 43
Table of Contents cont.

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>XERR</td>
<td>45</td>
</tr>
<tr>
<td>XEXC</td>
<td>46</td>
</tr>
<tr>
<td>XEXT</td>
<td>48</td>
</tr>
<tr>
<td>XEXZ</td>
<td>49</td>
</tr>
<tr>
<td>XFAC</td>
<td>50</td>
</tr>
<tr>
<td>XFBA</td>
<td>51</td>
</tr>
<tr>
<td>XFFN</td>
<td>52</td>
</tr>
<tr>
<td>XFTD</td>
<td>53</td>
</tr>
<tr>
<td>XFUM</td>
<td>54</td>
</tr>
<tr>
<td>XGCB</td>
<td>55</td>
</tr>
<tr>
<td>XGCD</td>
<td>56</td>
</tr>
<tr>
<td>XGCP</td>
<td>57</td>
</tr>
<tr>
<td>XGCR</td>
<td>58</td>
</tr>
<tr>
<td>XGLB</td>
<td>59</td>
</tr>
<tr>
<td>XGLM</td>
<td>61</td>
</tr>
<tr>
<td>XGLU</td>
<td>62</td>
</tr>
<tr>
<td>XGML</td>
<td>64</td>
</tr>
<tr>
<td>XGMP</td>
<td>65</td>
</tr>
<tr>
<td>XGNP</td>
<td>66</td>
</tr>
<tr>
<td>XGTM</td>
<td>68</td>
</tr>
<tr>
<td>XGUM</td>
<td>69</td>
</tr>
<tr>
<td>XISE</td>
<td>70</td>
</tr>
<tr>
<td>XKTBI</td>
<td>71</td>
</tr>
<tr>
<td>XKTMI</td>
<td>72</td>
</tr>
<tr>
<td>XLDF</td>
<td>73</td>
</tr>
<tr>
<td>XLER</td>
<td>75</td>
</tr>
<tr>
<td>XLFN</td>
<td>76</td>
</tr>
<tr>
<td>XLKF</td>
<td>78</td>
</tr>
<tr>
<td>XLKT</td>
<td>79</td>
</tr>
<tr>
<td>XLSR</td>
<td>80</td>
</tr>
<tr>
<td>XLSI</td>
<td>81</td>
</tr>
<tr>
<td>XLSI</td>
<td>81</td>
</tr>
<tr>
<td>XNOP</td>
<td>82</td>
</tr>
<tr>
<td>XPAD</td>
<td>84</td>
</tr>
<tr>
<td>XPBC</td>
<td>85</td>
</tr>
<tr>
<td>XPCB</td>
<td>86</td>
</tr>
<tr>
<td>XPCB</td>
<td>86</td>
</tr>
<tr>
<td>XPCB</td>
<td>86</td>
</tr>
<tr>
<td>XPCB</td>
<td>86</td>
</tr>
<tr>
<td>XPCC</td>
<td>87</td>
</tr>
<tr>
<td>XPCO</td>
<td>88</td>
</tr>
<tr>
<td>XPCR</td>
<td>89</td>
</tr>
<tr>
<td>XPCR</td>
<td>90</td>
</tr>
<tr>
<td>XPDC</td>
<td>91</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>XPEL</td>
<td>92</td>
</tr>
<tr>
<td>XPEM</td>
<td>94</td>
</tr>
<tr>
<td>XPLC</td>
<td>95</td>
</tr>
<tr>
<td>XPMC</td>
<td>96</td>
</tr>
<tr>
<td>XPSC</td>
<td>97</td>
</tr>
<tr>
<td>XPSF</td>
<td>99</td>
</tr>
<tr>
<td>XPSP</td>
<td>100</td>
</tr>
<tr>
<td>XRBF</td>
<td>101</td>
</tr>
<tr>
<td>XRCN</td>
<td>102</td>
</tr>
<tr>
<td>XRCP</td>
<td>103</td>
</tr>
<tr>
<td>XRDE</td>
<td>104</td>
</tr>
<tr>
<td>XRDM</td>
<td>105</td>
</tr>
<tr>
<td>XRDN</td>
<td>106</td>
</tr>
<tr>
<td>XRDT</td>
<td>107</td>
</tr>
<tr>
<td>XRFA</td>
<td>108</td>
</tr>
<tr>
<td>XRFP</td>
<td>109</td>
</tr>
<tr>
<td>XRLF</td>
<td>110</td>
</tr>
<tr>
<td>XRNF</td>
<td>111</td>
</tr>
<tr>
<td>XROO</td>
<td>112</td>
</tr>
<tr>
<td>XROP</td>
<td>113</td>
</tr>
<tr>
<td>XRPS</td>
<td>114</td>
</tr>
<tr>
<td>XRSE</td>
<td>115</td>
</tr>
<tr>
<td>XRSR</td>
<td>116</td>
</tr>
<tr>
<td>XRST</td>
<td>117</td>
</tr>
<tr>
<td>XRSZ</td>
<td>118</td>
</tr>
<tr>
<td>XRTE</td>
<td>119</td>
</tr>
<tr>
<td>XRTM</td>
<td>120</td>
</tr>
<tr>
<td>XRTP</td>
<td>121</td>
</tr>
<tr>
<td>XRTS</td>
<td>122</td>
</tr>
<tr>
<td>XRWF</td>
<td>123</td>
</tr>
<tr>
<td>XSEF</td>
<td>124</td>
</tr>
<tr>
<td>XSEV</td>
<td>126</td>
</tr>
<tr>
<td>XSMP</td>
<td>128</td>
</tr>
<tr>
<td>XSOE</td>
<td>129</td>
</tr>
<tr>
<td>XSOP</td>
<td>130</td>
</tr>
<tr>
<td>XSPF</td>
<td>132</td>
</tr>
<tr>
<td>XSTM</td>
<td>133</td>
</tr>
<tr>
<td>XSTP</td>
<td>134</td>
</tr>
<tr>
<td>XSUI</td>
<td>135</td>
</tr>
<tr>
<td>XSUP</td>
<td>137</td>
</tr>
</tbody>
</table>
Table of Contents cont.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSWP</td>
<td></td>
<td>138</td>
</tr>
<tr>
<td>XSZF</td>
<td></td>
<td>139</td>
</tr>
<tr>
<td>XTAB</td>
<td></td>
<td>141</td>
</tr>
<tr>
<td>XTEF</td>
<td></td>
<td>142</td>
</tr>
<tr>
<td>XTLP</td>
<td></td>
<td>143</td>
</tr>
<tr>
<td>XUAD</td>
<td></td>
<td>145</td>
</tr>
<tr>
<td>XUDT</td>
<td></td>
<td>146</td>
</tr>
<tr>
<td>XULF</td>
<td></td>
<td>147</td>
</tr>
<tr>
<td>XULT</td>
<td></td>
<td>148</td>
</tr>
<tr>
<td>XUSP</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>XUTM</td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>XVEC</td>
<td></td>
<td>151</td>
</tr>
<tr>
<td>XWBF</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>XWDT</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>XWFA</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>XWFP</td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>XWLF</td>
<td></td>
<td>156</td>
</tr>
<tr>
<td>XWSE</td>
<td></td>
<td>157</td>
</tr>
<tr>
<td>XWTM</td>
<td></td>
<td>158</td>
</tr>
<tr>
<td>XZFL</td>
<td></td>
<td>159</td>
</tr>
</tbody>
</table>
INTRODUCTION

This manual is a comprehensive reference to the PDOS assembly primitives. It is intended as a reference guide only, not as an introduction to assembly language programming. Some guidelines are given in this manual, however, for 68000 assembly programming with PDOS.

The PDOS assembly primitives are described separately in alphabetic order and make up the bulk of this manual. Also included in this manual is a list of calls divided by groups and a table of error codes.

Each assembly primitive description lists the value, the module, the syntax, and the registers of that call. It also describes how the call works and gives an example of that call used in an assembly language program. Possible errors, references to related calls, and other notes are also given. Examples are enclosed in a box and appear in a different typeface from the rest of the text. User input is bolded and comments are italicized. Keys are shown as bolded characters; for example, Ctrl C indicates that the “C” key is pressed while the “Control” key is being held down. Esc indicates the “Escape” key should be pressed. The / symbol indicates a carriage return and the \ symbol indicates a line feed.

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.

Guidelines For 68000 Assembly Programming

The following guidelines should prove useful to you in assembly programming for the PDOS system:


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

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

The XGML primitive may be used to reinitialize registers A5 and A6.
Guidelines for PDOS Assembly Programming

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

For example:

*Use BSRs instead of JSRs.*

<table>
<thead>
<tr>
<th>Good</th>
<th>Not Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR.L</td>
<td>SUBRT</td>
</tr>
<tr>
<td>SUBRT</td>
<td>JSR</td>
</tr>
</tbody>
</table>

*Use (PC) instead of absolute.*

```
LEA.L LAB(PC),A0  MOVEA.L #LAB,A0
...
LAB EQU *        LAB EQU *
```

*Set up OFFSET area.*

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

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

- XEXT
- XSOP

The primitives may also be specified as DC.W constants if you are using assembler other than the PDOS assembler.

```
DC.W $A00E ;XEXT
DC.W $A0EC ;XSOP
```

System Variables. The PDOS assembler supplies most system constants you are likely to require. These constants are supplied on reference after the "OPT PDOS" directive is executed. The following is the standard convention adopted for external PDOS symbols:

- xxx$ = TCB index (A6)
- xxx. = SYRAM constant
- xxxx. = SYRAM index (A5)
- .xxx = Global system constant
- m.xxx = Module constant
- m$xxx = Module entry point
- m_xxx = Module index
- xxx_ = User index
- MOVE.B U1PS(A6),D0
- MULU.W #TBZ.,D0
- MOVE.L TICS.(A5),D1
- MOVE.W #BPS,D7
- MOVE.W #B.PTMSK,SR
- BSR.L K2SPINT
- CLR.W B_TP$S(A0)
- ADDA.L AVL_-(A4),A0
The following illustrates how some of these constants might be used:

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

- **MOVE.AL**
  - `MAIL.(A5),A0`
  - *Point to the MAIL array*

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

- **ST.B**
  - `DFLG.(A5)`
  - *Set hard partitioned directory*

- **ST.B**
  - `TLCK.(A5)`
  - *Lock current task*

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

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

- **ST.B**
  - `ECFS(A6)`
  - *Disable console echo*

- **MOVE.AL**
  - `BIOS.(A5),A0`
  - *Read system ID characters*

- **MOVE.W**
  - `B_SID(AO),DO`
  - *Assembly Format. PDOS assembly text has the following conventions:*

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

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

- **Source file documentation. PDOS source files have the following conventions:***

  a. Assembler TTL directive
  b. File name followed by last update date

```
TTL FILE - PDOS PROGRAM FILE
  FILE:SR 07/22/87
***********************************************************************
*  FFFFFF IIII LL EEEEE  *
*  FF II LL EE  *
*  FF II LL EE  *
*  FFFFFF II LL EEEEE  *
*  FF II LL EE  *
*  FF II LL EE  *
*  FF IIII LLLLLL EEEEE  *
*  *
***********************************************************************
```

c. Company identification with copyright notices

```
* Eyring Research Institute Inc.
* Copyright 1983-87
* ALL RIGHTS RESERVED
*=
```
Guidelines for PDOS Assembly Programming

d. Module identification
c. Author of program
f. Who authorizes any changes
g. Revision history

| * | Module Name: FILE |
| * | Author: John Doe |
| * | Changes Authorized by: |
| * | Revision History: |
| * | DATE | R.V | DESCRIPTION |
| * | 07/08/87 | 2.36 | D$INT called from XCTB |
| * | 07/18/87 | 2.37 | XLER enables echo ECF$ |
| * | 07/22/87 | 2.38 | Reset event |

h. Program ID

| * | FILE | IDNT | 2.38 | M68000 PDOS |
| * | *************************************** |

PDOS Assembly Language Calls

PDOS assembly primitives are one word A-line instructions which normally use the exception vector at memory location $00000028. Most primitives use 68000 registers to pass parameters to and results from resident PDOS routines. Registers for system calls are generally used from D0 up and A0 up. Some calls (XPMC, XTAB, and XDMP) pass the relative address to the call by placing the address word immediately following the call. Status returns are used after the call. Some primitives return an error in the status register while other primitives return a status depending on the state of the primitive. For example, the XGCB (conditional get character) primitive returns one the following conditions in the status register: EQ - no character; LO - Ctrl C; LT - Esc; MI - Ctrl C or Esc.

```
LOOP XGCB  ;CHARACTER?
    BEQ.S NONE  ;N
    BLO.S QUIT  ;Y, C, DONE
    BLT.S NEXT  ;CONTINUE
    CMPI.B #'0',DO  ;NUMBER
```

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. D0 holds the error code and the status is either NE for no error code or the error code itself. The following example demonstrates 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 #
```
Guidelines for PDOS Assembly Programming

The following illustrates how some of these constants might be used:

- **BSET.B** #~118,118/8+EVTB.(A5) Set event 118
- **MOVEA.L** MAIL.(A5),A0 Point to the MAIL array
- **MOVE.L** TICS.(A5),D1 Read system tics
- **ST.B** DFLG.(A5) Set hard partitioned directory
- **ST.B** TLCK.(A5) Lock current task
- **MOVE.B** #2,PRT$(A6) Set input port #
- **MOVE.B** #5,FEC$(A6) Set file expansion count
- **ST.B** ECF$(A6) Disable console echo
- **MOVEA.L** BIOS.(A5),A0 Read system ID characters
- **MOVE.W** B_SID(A0),D0

**Assembly Format.** PDOS assembly text has the following conventions:

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

```
*            LABEL   CMPI.W  #10,D1 ; LESS THAN 107
              BLT.S   LABEL ; Y
```

**Source file documentation.** PDOS source files have the following conventions:

- a. Assembler TTL directive
- b. File name followed by last update date

```
TTL      FILE - PDOS PROGRAM FILE
         *      FILE:SR     07/22/87
         *************************************************
         *            FFFFFF   IIII   LL     EE     EEEEEE  *
         *            FF       II     LL     EE     *     *
         *            FF       II     LL     EE     *     *
         *            FFFFFF   II     LL     EE     EEEEEE  *
         *            FF       II     LL     EE     *     *
         *            FF       II     LL     EE     *     *
         *            FF       IIII   LLLLLL   EEEEEE  *
         *            *            *************************************************
```

c. Company identification with copyright notices

```
*            Eyring Research Institute Inc.  *
*            Copyright 1983-87          *
*            ALL RIGHTS RESERVED       *
*            *************************************************
```
Guidelines for PDOS Assembly Programming

d. Module identification
e. Author of program
f. Who authorizes any changes
g. Revision history

*= Module Name: FILE
*= Author: John Doe
*= Changes Authorized by:
*= Revision History:
*= DATE R.V DESCRIPTION
*= 07/08/87 2.36 D$INT called from XCTB
*= 07/18/87 2.37 XLER enables echo ECF$
*= 07/22/87 2.38 Reset event

h. Program ID

FILE IDNT 2.38 M68000 PDOS
=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*

PDOS Assembly Language Calls

PDOS assembly primitives are one word A-line instructions which normally use the exception vector at memory location $00000028. Most primitives use 68000 registers to pass parameters to and results from resident PDOS routines. Registers for system calls are generally used from DO up and AO up. Some calls (XPMC, XTAB, and XDMP) pass the relative address to the call by placing the address word immediately following the call. Status returns are used after the call. Some primitives return an error in the status register while other primitives return a status depending on the state of the primitive. For example, the XGCB (conditional get character) primitive returns one of the following conditions in the status register: EQ - no character; LO - Ctrl C; LT - Esc; MI - Ctrl C or Esc.

```
LOOP XGCB ;CHARACTER?
  BEQ.S NONE ;N
  BLO.S QUIT ;Y, 'C, DONE
  BLT.S NEXT ;CONTINUE
  CMP.B #0',DO ;NUMBER
```

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. DO holds the error code and the status is either NE for no error code or the error code itself. The following example demonstrates 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,SLT(A4) ;N, SAVE SLOT #
```
System Support Calls cont.

XPAD - Pack ASCII date
XUAD - Unpack ASCII Date
XUDT - Unpack date
XUTM - Unpack time
XWDT - Write date
XWTM - Write time
XGNP - Get next parameter

File Support Calls

File support calls augment the file manager. 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
XLST - List file directory
XBFL - Build file directory list
XRDE - Read next directory entry
XRDN - Read directory entry by name
XAPF - Append file
XCPY - Copy file
XCHF - Chain file
XLDF - Load file
XRCN - Reset console inputs
XRST - Reset disk
XSZF - Get disk size

File Management Calls

The file management calls of PDOS 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 file
XROO - Open random read only file
XROP - Open random file
XNOP - Open shared random file
XLKF - Lock file
XULF - Unlock file
XRFP - Read file position
XRWF - Rewind file
PDOS Assembly Language Calls

File Management Calls cont.

- 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 with attribute
- XCLF - Close file

Disk Access Calls

Disk access calls use 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
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Bad File Name</td>
</tr>
<tr>
<td>51</td>
<td>File Already Defined</td>
</tr>
<tr>
<td>52</td>
<td>File Not Open</td>
</tr>
<tr>
<td>53</td>
<td>File Not Defined</td>
</tr>
<tr>
<td>54</td>
<td>Bad File Attribute</td>
</tr>
<tr>
<td>55</td>
<td>Too Few Contiguous</td>
</tr>
<tr>
<td>56</td>
<td>End of File</td>
</tr>
<tr>
<td>57</td>
<td>Directory Full</td>
</tr>
<tr>
<td>58</td>
<td>File Writ/Del Prot</td>
</tr>
<tr>
<td>59</td>
<td>Bad File Slot</td>
</tr>
<tr>
<td>60</td>
<td>File Space Full</td>
</tr>
<tr>
<td>61</td>
<td>File Already Open</td>
</tr>
<tr>
<td>62</td>
<td>Bad Message Ptr Call</td>
</tr>
<tr>
<td>63</td>
<td>Bad Object Tag</td>
</tr>
<tr>
<td>64</td>
<td>Not Executable</td>
</tr>
<tr>
<td>65</td>
<td>Bad Port/Baud Rate</td>
</tr>
<tr>
<td>66</td>
<td>Bad Parameter</td>
</tr>
<tr>
<td>67</td>
<td>Not PDOS Disk</td>
</tr>
<tr>
<td>68</td>
<td>Out of File Slots</td>
</tr>
<tr>
<td>70</td>
<td>Position &gt; EOF</td>
</tr>
<tr>
<td>71</td>
<td>AC File Nesting &gt; 2</td>
</tr>
<tr>
<td>72</td>
<td>Too Many Tasks</td>
</tr>
<tr>
<td>73</td>
<td>Not Enough Memory</td>
</tr>
<tr>
<td>74</td>
<td>Non-existent Task</td>
</tr>
<tr>
<td>75</td>
<td>File Locked</td>
</tr>
<tr>
<td>76</td>
<td>Not Memory Resident</td>
</tr>
<tr>
<td>77</td>
<td>Msg Buffer Full</td>
</tr>
<tr>
<td>78</td>
<td>Bad Memory Address</td>
</tr>
<tr>
<td>79</td>
<td>Bad Driver Call</td>
</tr>
<tr>
<td>80</td>
<td>Delay Queue Full</td>
</tr>
<tr>
<td>85</td>
<td>Task Abort</td>
</tr>
<tr>
<td>86</td>
<td>Suspend on Port 0</td>
</tr>
<tr>
<td>87</td>
<td>Exception</td>
</tr>
</tbody>
</table>
The following section describes each assembly call in alphabetical order. The description includes its syntax, the PDOS module in which it is found, possible errors, and an example demonstrating how the call may be used.
Value: $A006
Module: MPDOSK1
Syntax: X881
Registers: None

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 the PDOS monitor.

See Also: BIOS in *PDOS Developer's Reference Manual*
Possible Errors: None

Example:

```
START X881
FMOVE.L #100,FPO
FDIV.W #3,FPO
```
XAPF
Append File

Value: $A0AA
Module: MPDOSF
Syntax: XAPF
<status error return>
Registers: In (A1) = Source file name (A2) = Destination file name

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.

Possible Errors:
-1 = Break
50 = Bad File Name
53 = File Not Defined
60 = File Space Full
61 = File Already Open
68 = Not PDOS Disk
69 = Out of File Slots
Disk errors

Example:

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

SF1 DC.B 'FILE1',0
SF2 DC.B 'FILE2',0
EVEN
```
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.

\[ F8BT. = FWPI \ 8DCS \]
\[ \\
\\\\\\\\\\\\\_ 0 = Ctrl S Ctrl Q enable \\
\\\\\\\\\\\\\_ 1 = Ignore control character \\
\\\\\\\\\\\\\_ 2 = DTR enable \\
\\\\\\\\\\\\\_ 3 = 8-bit character enable \\
\\\\\\\\\\\\\_ 4 = Receiver interrupts disable \\
\\\\\\\\\\\\\_ 5 = Even parity enable \\
\\\\\\\\\\\\\_ 6 = *Reserved \\
\\\\\\\\\\\\\_ 7 = **Reserved \\
\]

*Used to clear all bits

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 Ctrl S Ctrl Q and/or DTR handshaking, receiver parity and interrupt disable, 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 8 or the corresponding baud rates of 19200, 9600, 4800, 2400, 1200, 600, 300, 110, 38400. If register D3 is equal to -1, then only port 2 is set. 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.
XBCP - Baud Console Port

D3.W = Baud =
0 = 19200 baud
1 = 9600 baud
2 = 4800 baud
3 = 2400 baud
4 = 1200 baud
5 = 600 baud
6 = 300 baud
7 = 110 baud
8 = 38400 baud

Baud rate 38400 is not supported by all BIOSes.

See Also:
XRPS - Read Port Status
XSPF - Set Port Flag

Possible Errors:
66 = Bad Port/Baud Rate

Example:

```
START MOVE.W #103,D2 ;PORT 3 W/"S"Q
MOVE.W #19200,D3 ;19.2K BAUD
MOVEQ.L #0,D4 ;NO TYPE CHANGE
XBCP ;BAUD PORT
BNE.S ERROR
.....
```
Value: $A0B8
Module: MPDOOSM
Syntax: XBFL
<status error return>

Registers:
In  
(A1) = List specifications
(A2) = Beginning buffer address
(A3) = End buffer address
Out  
(A3) = Updated buffer end address

Description:
The BUILD FILE DIRECTORY LIST primitive builds a serial list of file names in memory as selected by the list specifications. Address register A1 points to the file list specifications.

List specifications:

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

where

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

Address registers A2 and A3 point to the beginning and end of the memory buffer respectively. Register A3 is updated to a word boundary just after the last file name null.

Possible Errors:

- Disk errors
  - 67 = Bad Parameter
  - 73 = Not Enough Memory
**Example:**

```plaintext
GETL   LEA.L SPC(PC),A1 ;POINT TO LIST
LEA.L BUF(PC),A2 ;GET BUFFER ADDRESS
LEA.L EBUF(PC),A3 ;GET END POINTER
XBFL ;BUILD LIST
         BNE.S ERROR
* PRNT  TST.B (A1) ;ENTRY?
         BEQ.S DONE ;N
         XPCL ;Y, OUTPUT CRLF
         XPLC ;OUTPUT ENTRY
* NEXT  TST.B (A1)+ ;NEXT, DONE?
         BNE.S NEXT ;N
         BRA.S PRNT ;Y
*
DONE . . .
*
ERROR . . .

SPC  DC.B '@:SR;%0',0
BUF  DS.B 500
EBUF  EQU *
```
The DEBUG CALL primitive breaks from the user program and enters the PDOS debugger. All registers are saved and you are prompted for additional commands. The following are legal debugger commands for the resident debugger:

- **A0-7** A-reg
- **B(#,a)** Lst/def break
- **D0-7** D-reg
- **{#}G** Go & break
- **H** Help message
- **M** Last dump
- **N#** O=Wrd,1=Byt,4=Long 5=Byt skp, +2=w/o read
- **O** Offset
- **P** PC
- **Q** Exit
- **R** Reg dump
- **S** Status
- **T** Trace
- **U** Unit
- **V** Control IAC
- **W(s,e)** Window
- **X** Set breaks & exit
- **Z** Reset

### Trace Options:

- **^D** Disassemble
- **-** Open previous
- **LF** Open next
- **#** Mem IAC
- **##** Mem dump
- **##+** Disassemble
- **##,(WL)** Find B/W/L
- **#(0-7)** d(Ax)
- **++** # + offset

If you use the SMARTBUG debugger, refer to SMARTBUG Reference Manual for valid commands.

**See Also:**

- XDMP - Dump Memory From Stack
- XRDM - Dump Registers
- PB - PDOS Debugger *(PDOS Monitor, Editor, Utilities manual)*

**SMARTBUG Reference Manual**

**Possible Errors:**

None
Example:

```
****
XCBC  ;BREAK?
BLO.S CONTC  ;Y, ^C
BLT.S ESCAP  ;Y, ESC
BRA.S LOOP   ;N, CONTINUE
*
CONTC ****  ;CONTROL C
*
BRA.S BEGIN  ;START AGAIN
*
ESCAP XPMC  BRKM  ;OUTPUT '>>BREAK'
XEXT         ;EXIT TO PDOS
*
BRKM DC.B  $0A,$0D ;BREAK MESSAGE
DC.B '>>BREAK',0
```
## XCBC

**Check For Break Character**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A072</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XCBC</td>
</tr>
<tr>
<td></td>
<td>&lt;status return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>Out SR =</td>
</tr>
<tr>
<td></td>
<td>EQ....No break</td>
</tr>
<tr>
<td></td>
<td>LO....Ctrl C, Clear flag &amp; buffer</td>
</tr>
<tr>
<td></td>
<td>LT....Esc, Clear flag</td>
</tr>
<tr>
<td></td>
<td>MI....Ctrl C or Esc</td>
</tr>
</tbody>
</table>

If the ignore control character bit ($02) of the port flag is set, then XCBC always returns .EQ. status.

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 “AC” 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.)

Possible Errors: None
XCBD

Convert Binary to Decimal

Value: $A050
Module: MPDOSK3
Syntax: 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 (MWBS). 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:
XCBX - Convert To Decimal In Buffer

Possible Errors:
None

Example:

MOVE.L #1234, D1 ;GET NUMBER
XCBD ;CONVERT TO PRINT
XPLC ;PRINT
....

******************************************************************************
* OUTPUT LEFT JUSTIFIED NUMBER *
* *
*     D0.W = # OF PLACES
*     D1.L = NUMBER
*
* LEFT MOVEM.L D0/A0-A1,-(A7)
*   XCBD ;CONVERT
*   MOVEA.L A1,A0 ;GET POINTER
*
* LEFT02 SUBQ.W #1,D0 ;COUNT LENGTH
*   TST.B (A0)+ ;END?
*   BNE.S LEFT02 ;N
*
* LEFT04 XPSP ;OUTPUT SPACE
*   SUBQ.W #1,D0 ;DONE?
*   BPL.S LEFT04 ;N
*   XPLC ;Y, OUTPUT #
*   MOVEM.L (A7)+,D0/A0-A1
*   RTS

20  3.3 - 10/87  PDOS ASSEMBLY PRIMITIVES REFERENCE
The CONVERT BINARY TO HEX primitive converts a 32-bit number to its hexadecimal (base 16) representation. The number is passed in data register DI and a pointer to the ASCII string is returned in address register A1. The converted string is found in the monitor work buffer (MWBS$) of the task control block and consists of eight hexadecimal characters followed by a null.

**Example:**

```
MOVEQ.L #123,D1 ;GET NUMBER
XCBH ;GET HEX CONVERSION
MOVEQ.L #'$',D0 ;ADD HEX SIGN
XPCC ;PRINT
XPIC ;PRINT 8 HEX CHARACTERS
....

******************************************************************************
* DUMP REGISTERS ON USER STACK *
* *
* USP = A7 = RETURN PC *
* DO-D7 *
* A0-A7 *
* DMRG02 XPCC ;OUT CRLF
  XPLC ;OUT LINE TYPE
  MOVE.W #' : ',D0
*
  DMRG04 XPCC ;OUT DELIMITER
  MOVE.L (A7)+,D1 ;GET REGISTER
  XCBH ;CONVERT
  XPLC ;OUTPUT
  MOVEQ.L #' ',D0 ;CHANGE TO ' '
  LSR.L #1,D4 ;4 DONE?
  BCS.S DMRG04 ;N
  XPCC ;Y, OUT SPACE
  LSR.L #1,D4 ;CRLF?
  BCS.S DMRG04 ;N
  MOVE.W #'OA*',D0 ;Y, CHANGE TO 'A'
  LSR.L #1,D4 ;MORE?
  BCS.S DMRG02 ;Y
  JMP (A0) ;N, RETURN
```

XCHX - Convert Binary To Hex In Buffer

None
**XCBM**

**Convert to Decimal with Message**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A054</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK3</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XCBM &lt;message&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>![Registers Diagram]</td>
</tr>
<tr>
<td>Description:</td>
<td>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 (MWBS) 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.</td>
</tr>
</tbody>
</table>

**Possible Errors:** None

**Example:**

```
START MOVE.L #$80000004,D1
  *  LOOP XPMC MES1 ;HEADING
    XCBH ;CONVERT HEX
    XPLC XCBM MES2 ;CONVERT DECIMAL
    XPLC SUBQ.L #1,D1
    CMPI.L #$7FFFFFFC,D1
    BHS.S LOOP
  XEXT MES1 DC.B $0A,$0D,'Hex $',0
  MES2 DC.B '=' ',0
  EVEN
END START
```

&times;TEST

<table>
<thead>
<tr>
<th>Hex</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$80000004</td>
<td>-2147483644</td>
</tr>
<tr>
<td>$80000003</td>
<td>-2147483645</td>
</tr>
<tr>
<td>$80000002</td>
<td>-2147483646</td>
</tr>
<tr>
<td>$80000001</td>
<td>-2147483647</td>
</tr>
<tr>
<td>$80000000</td>
<td>-2147483648</td>
</tr>
<tr>
<td>$7FFFFFFF</td>
<td>2147483647</td>
</tr>
<tr>
<td>$7FFFFFFE</td>
<td>2147483646</td>
</tr>
<tr>
<td>$7FFFFFFD</td>
<td>2147483645</td>
</tr>
<tr>
<td>$7FFFFFFC</td>
<td>2147483644</td>
</tr>
</tbody>
</table>

*x*
### XCBP

**Check for Break or Pause**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A074</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XCBP</td>
</tr>
<tr>
<td></td>
<td>&lt;status return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>Out SR =</td>
</tr>
<tr>
<td></td>
<td>EQ...No character</td>
</tr>
<tr>
<td></td>
<td>LT...Esc</td>
</tr>
<tr>
<td></td>
<td>LO...Ctrl C</td>
</tr>
<tr>
<td></td>
<td>NE...Pause</td>
</tr>
</tbody>
</table>

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:

```
    ;J'Strike any key...';J
```

A Ctrl C will abort any assigned console file and return the status "LO". If a "BLT" instruction follows the XCBP primitive and an Esc character is entered, then the call returns with status "LT". Otherwise, an Esc will abort your program to the PDOS monitor. An "EQ" status indicates that no character was entered. An "NE" status indicates a pause has occurred.

**Possible Errors:**

None

**Example:**

```
LOOP     ....        ;OUTPUT
          
XCBP     ;LOOK FOR PAUSE
BL.T.S EXIT ;ESC
          
BRA.S LOOP ;CONTINUE
          *

EXIT     ....        ;ESC
```
XCBX
Convert to Decimal in Buffer

Value: $A06A
Module: MPDOSK3
Syntax: 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: XCBD - Convert Binary To Decimal

Possible Errors: None

Example:

```
MOVEA.L A6,A1 ;POINT TO USER BUF
MOVEQ.L #12,D1 ;GET #
BSR.S OUTS ;OUTPUT TO BUFFER
XPBC ;OUTPUT BUFFER
....
OUTS XCBX ;CONVERT #
*
OUTS02 TST.B (A1)+ ;END?
BNE.S OUTS02 ;N
SUBQ.W #1,A1 ;Y, BACKUP
RTS ;RETURN
```
XCDB
Convert ASCII to Binary

Value:  $A056

Module:  MPDOSK3

Syntax:  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....#

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 A1 has the address of the next character.

Possible Errors:
None

Example:

```
START  MOVEQ.L #0,D5  ;GET DEFAULT
XPMC  MES1  ;OUTPUT PROMPT
XGLU  ;GET REPLY
  BLS.S STRT04  ;USE DEFAULT
Xcdb  ;CONVERT, OK?
  BGT.S STRT02  ;Y
XPMC  ERM1  ;N, REPORT
BRA.S START  ;TRY AGAIN

*  STRT02  MOVE.L D1,D5  ;SAVE VALUE

  STRT04

MES1  DC.B  $0A,$0D,'ANSWER=',0
ERML  DC.B  $0A,$0D,'INVALID!',0
```

PDOS ASSEMBLY PRIMITIVES REFERENCE  3.3 - 10/87  25
## XCFA

### Close File with Attribute

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A0D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSF</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XCFA</td>
</tr>
<tr>
<td>Description:</td>
<td>The <strong>CLOSE FILE WITH ATTRIBUTES</strong> 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.</td>
</tr>
</tbody>
</table>

#### Registers:

- **D1.W** = File ID
- **D2.B** = New attribute

#### Syntax Example:

- **D2.B** = $80 AC or Procedure file
- **D2.B** = $40 BN or Binary file
- **D2.B** = $20 OB or 68000 object file
- **D2.B** = $10 SY or 68000 memory image
- **D2.B** = $08 BX or BASIC binary token file
- **D2.B** = $04 EX or BASIC ASCII file
- **D2.B** = $02 TX or Text file
- **D2.B** = $01 DR or System I/O driver
- **D2.B** = $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.

If the file is not altered, then XCFA will not alter the file attributes.

**D1.W** = File ID = (Disk #) x 256 + (File slot index)

### See Also:

- XRFA - Read File Attributes
- XWFA - Write File Attributes
- XWFP - Write File Parameters

### Possible Errors:

- 52 = File Not Open
- 59 = Bad File Slot
- 75 = File Locked
- Disk errors
Example:

```
MOVE.W D5,D1 ;GET FILE ID
MOVE.B #$20,D2 ;CLOSE AS OBJECT
XCFA ;CLOSE FILE
BNE.S ERROR
.....
```
XCHF

Chain File

Value: $A0AC
Module: MPDOSM
Syntax: XCHF
Registers:

In A1.L = File name

The primitive returns only on error.

Description:

The CHAIN FILE primitive is used by the PDOS monitor to execute program files. The primitive chains from one program to another according to the file type. Address register A1 points to the chain file name. The file type determines how the file is to be executed.

If the file is typed “OB” or “SY”, then the 68000 loader is called (XLDF). If the file is typed “BX” or “EX”, then the PDOS BASIC interpreter loads the file and begins executing at the lowest line number. Likewise, if the file is typed “AC”, then control returns back to the PDOS monitor and further requests for console characters reference the file.

The XCHF call returns only if an error occurs during the chain operation. All other errors, such as those occurring in BASIC, return to the PDOS monitor. Parameters may be passed from one program to another through the user TEMP variables located in the task control block or through the system messages buffers.

See Also:

XEXZ - Exit To Monitor With Command

Possible Errors:

50 = Bad File Name
53 = File Not Defined
60 = File Space Full
63 = Bad Object Tag
65 = Not Executable
77 = Not Memory Resident
Disk errors

Example:

```
LEA.L FILEN(PC), A1 ; GET FILE NAME
XCHF ; CHAIN FILE
XERR ; PROBLEM
*
FILEN DC.B 'NEXTPRGM', 0
EVEN
```
### XCHX

**Convert Binary to Hex in Buffer**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A068</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK3</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XCHX</td>
</tr>
<tr>
<td>Registers:</td>
<td>In D1.L = Number (A1) = Output buffer</td>
</tr>
<tr>
<td>Description:</td>
<td>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.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XCBH - Convert Binary To Hex</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>None</td>
</tr>
</tbody>
</table>
XCHX - Convert Binary to Hex in Buffer

Example:

START MOVE.L #$80000004,D1

LOOP MOVEA.L A6,A1 ;USER BUFFER
     BSR.S OUTS ;OUT HEADING
     DC.W MES1-*
     XCHX ;CONVERT HEX

LOOP2 TST.B (A1)+ ;END?
     BNE.S LOOP2 ;N
     SUBQ.W #1,A1 ;Y
     BSR.S OUTS ;' = '
     DC.W MES2-*
     XCHX ;CONVERT DECIMAL

LOOP4 TST.B (A1)+ ;END?
     BNE.S LOOP4 ;N
     XFB ;Y, OUTPUT
     SUBQ.L #1,D1
     CMP.L #$7FFFFFFF,D1
     BS.S LOOP
     XEXT

OUTS MOVEA.L (A7),A0 ;GET ADDRESS
     ADDQ.L #2,(A7) ;ADJUST PC
     ADDA.W (A0)+,A0

OUTS2 MOVE.B (A0)+,(A1)+
     BNE.S OUTS2
     SUBQ.W #1,A1
     RTS

MES1 DC.B $0A,$0D,'Hex $',0
MES2 DC.B ' = ',0
EVEN
END START

x>TEST
Hex $80000004 = -2147483644
Hex $80000003 = -2147483645
Hex $80000002 = -2147483646
Hex $80000001 = -2147483647
Hex $80000000 = -2147483648
Hex $7FFFFFFF = 2147483647
Hex $7FFFFFFE = 2147483646
Hex $7FFFFFFD = 2147483645
Hex $7FFFFFFC = 2147483644
x>
Value: $A0D2
Module: MPDOSF
Syntax: 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.
File ID = (Disk #) x 256 + (File slot index)
If the file was opened for random or shared access, then the END-OF-FILE marker is updated only if the file was extended (i.e. data was written after the current END-OF-FILE marker). If the file has been altered, the current date and time is stored in the LAST UPDATE variable of the file directory. All open files must be closed at or before the completion of a task (or before disks are removed from the system)! Otherwise, directory information is lost and possibly even the file itself.
Possible Errors:
52 = File Not Open
59 = Bad File Slot
75 = File Locked
Disk errors
Example:

```
MOVE.W D5,D1 ;GET FILE ID
XCLF ;CLOSE FILE
BNE.S ERROR
....
ERROR CLR.L D1
MOVE.W D0,D1 ;GET ERROR #
XCBM ERM1 ;CONVERT
XPLC ;OUTPUT
....
ERM1 DC.B $0A,$0D
DC.B 'PDOS CLOSE ERR ',0
EVEN
```
XCLS

Clear Screen

Value: $A076
Module: MPDOSK2
Syntax: XCLS
Registers: None

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

The CLEAR SCREEN primitive clears the console screen, homes the cursor, and clears the column counter. This function is adapted to the type of console terminals used in the PDOS system. The character sequence to clear the screen is located in the task control block variable CSC$(A6). These characters are transferred from the parent task to the spawned task during creation. The initial characters come from the BIOS module.

CSC$(A6) = E111 1111 E222 2222
\ \ \ 2nd character
\ \ \ 2nd Esc
\ \ \ \ \ \ 1st character
\ \ \ 1st Esc

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 or if the first byte equals $FF, then PDOS makes a call into the BIOS for custom clear screens. The entry point is B_CLS beyond the BIOS table. The MTERM utility normally maintains the CSC$ field, although it can be altered under program control. The initial definition of CSC$ is found in the MBIOS:SR file and can be modified by doing a new SYSGEN.

See Also: XRCP - Read Port Cursor Position
BIOS in PDOS Developer's Reference Manual

Possible Errors: None

Example:

```
XCLS ;CLEAR SCREEN
XPMC MES01 ;OUTPUT MESSAGE
```
XCPY
Copy File

Value: $A0AE
Module: MPDOSF
Syntax: XCPY
<status error return>

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

Description:
A Ctrl C terminates this primitive and returns the error -1 in register D0.
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 trans­
ferred to the destination file.

Possible Errors:
-1 = Break File Transfer
50 = Bad File Name
53 = File Not Defined
60 = File Space Full
61 = File Already Open
68 = Not PDOS Disk
69 = Out of File Slots
Disk errors

Example:

```
LEA.L FILES(PC),A1 ;SOURCE FILE NAME
LEA.L FILED(PC),A2 ;DEST. FILE NAME
XCPY ;COPY FILE
    BNE.S ERROR ;PROBLEM
     .... ;CONTINUE

FILES   DC.B 'TEMP',0
FILED   DC.B 'TEMP:bk/1',0
EVEN
```

PDOS ASSEMBLY PRIMITIVES REFERENCE 3.3 - 10/87
Create Task Block

Value: $A026

Module: MPDOSK1

Syntax: XCTB

<status error return>

Registers:

In

D0.W = Task size (1k byte increments)
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

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 1K bytes) is allocated to the new task. If there is not a block big enough, then the upper memory of the parent task is allocated to the new task. The parent task’s memory is then reduced by D0.W x 1K bytes. Address register A2 points to the new task command line. If A2 is zero, then the monitor is invoked.

Example:

```assembly
If D0>0 then:
  D0=Task size
  (A2)=Task command line
  (0=Monitor)
  MOVEQ.L #10,D0 ;10 K BYTES
  MOVEQ.L #64,D1 ;PRIORITY 64
  MOVEQ.L #1,D2 ;PORT 1
  SUBA.L A2,A2 ;CALL MONITOR
  XCTB ;CREATE TASK
  BNE.S ERROR
```

If register D0.W is zero, then registers A0 and A1 specify the new task’s memory limits. Register A2 specifies the task’s starting PC. The task control block begins at (A0) and is immediately followed by an XEXT primitive. The task user stack pointer is set at (A1). Thus, the new program should allow $502 bytes at the low end and enough user stack space at the upper end.
Example:

If D0=0 then: (A2)=Task entry address
A0-A1=New task memory limits

MOVEQ.L #0,D0 ;USE A0-A1 BOUNDS
MOVEQ.L #64,D1 ;PRIORITY 64
MOVEQ.L #1,D2 ;PORT 1
LEA.L SRAM,A0 ;TCB ADDR (START)
LEA.L ERAM,A1
LEA.L P(PC),A2 ;PC
XCTB ;CREATE TASK
BNE.S ERROR

If data register D0.W is negative, then registers A0 and A1 specify the new task’s memory limits. Register A2 points to the new task command line. (If A2=0, then the monitor is invoked.) The command line is transferred to the spawned program via a system message buffer. The maximum length of a command line is 64 characters. When the task is scheduled for the first time, the message buffers are searched for a command. Messages with a source task equal to $FF are considered commands and moved to the task’s monitor buffer. The task CLI then processes the line. If no command message is found, then the monitor is called directly.

Example:

If D0=<0 then: (A2)=Task command line
(O=Monitor)
A0-A1=New task memory limits

MOVEQ.L #0,D0 ;USE A0-A1 BOUNDS
MOVEQ.L #64,D1 ;PRIORITY 64
MOVEQ.L #1,D2 ;PORT 1
LEA.L SRAM,A0 ;TCB ADDR (START)
LEA.L ERAM,A1
LEA.L P(PC),A2 ;PC
XCTB ;CREATE TASK
BNE.S ERROR

C DC.B 'PRGM1',0

Data register D1.W specifies the new task’s priority. The range is from 1 to 255. The larger the number, the higher the priority.

D1=Task priority

Data register D2.W specifies the I/O port to be used by the new task. If register D2.W is positive, then the port is available for both input and output. If register D2.W is negative, then the port is used only for output. If register D2.W is zero, then no port is assigned. Only one task may be assigned to any 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 DSINT 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.
**XCTB - Create Task Block**

D2 = I/O port
- If D2 = 0, then phantom port (no I/O)
- If D2 > 0, then port is used for I/O
- If D2 < 0, then port is used for output only

If you specify the address as a file parameter, the system does not check to see if the memory is already allocated to another task. Use caution or it may crash your system.

Possible Errors:

- 72 = Too Many Tasks
- 73 = Not Enough Memory
**XDEV**

**Delay Set/Clear Event**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A032</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XDEV &lt;status error return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>In D0.L = Time D1.B = Logical Event (+=Set(1), -=Clear(0))</td>
</tr>
</tbody>
</table>

If D0.L=0, then the D1.B event is removed from the delay list.

**Description:**

The DELAY SET/CLEAR EVENT primitive places a logical timed event in a system delay list 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 cleared if negative. If the event already exists in the delay list, it is replaced by the new entry. If the time specified in D0 equals zero, then the event equal to D1.B is removed from the delay list. If D1.B is positive, event D1.B is first cleared. If D1.B is negative, event D1.B is set before placing the event in the delay list and exiting the primitive.

**See Also:**

- XSEF - Set Event Flag With Swap
- XSEV - Set Event Flag
- XSUI - Suspend Until Interrupt
- XTEF - Test Event Flag
- XDPE - Delay on Physical Event

**Possible Errors:**

- 83 = Delay Queue Full

**Example:**

```assembly
GETC XGCC ;CHARACTER?
  BNE.S GETC2 ;Y
  MOVEQ.L #100,D0 ;N, GET DELAY
  MOVE.L #128,D1 ;USER LOCAL EVENT
  XDEV ;DELAY 128 1 SECOND
  BNE.S GETC ;FULL
  LSL.W #8,D1 ;GET 128/(PORT+96)
  MOVE.B #96,D1
  ADD.B PRT$(A6),D1
  XSUI ;SUSPEND
  CMP.B D0,D1 ;CHARACTER EVENT?
  BEQ.S GETC ;Y
  XRTM ;N, READ TIME
  MOVE.B T(A6),D0 ;GET LAST CHARACTER
  CMP.B T(A6),D0 ;SAME TIME?
  BEQ.S GETC ;Y, TRY AGAIN
  MOVE.L (A1)+,T(A6) ;N, SAVE NEW TIME
  MOVE.L (A1),T+4(A6)
  CLR.B T+8(A6)
  BSR.S POSIT ;POSITION & OUTPUT TIME
  DC.W 23*256+11
  DC.W 0
  BRA.S GETC ;TRY AGAIN
```
XDFL

Define File

Value: $A0D4
Module: MPDOSF
Syntax: 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.

The filename convention is as follows where upper and lower case are unique:

APPPPPPP:PPP;NNN/NNN

# -- Auto-create flag may prefix filename
A -- Alpha characters A-Z or a-z
P -- Printable characters except ":", ";", "/". The "." character may be used, but will conflict with the monitor command separator unless the filename is enclosed within parentheses
N -- Number in the range of 0-255

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, the value stored in the SYRAM variable "FECT." + 1 is used to define the number of sectors that will be 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.

If the register D0 is non-zero, then the EOF pointer will be set to point at the end of the last allocated sector; otherwise, the EOF pointer will point at the beginning of the first allocated sector.

Possible Errors:

50 = Bad File Name
51 = File Already Defined
55 = Too Few Contiguous Sectors
57 = File Directory Full
61 = File Already Open
68 = Not PDOS Disk
Disk errors
Example:

```
CLR.L  DO  ;DEFAULT SIZE
LEA.L  FN(PC),A1  ;GET FILE NAME
XDFL   ;DEFINE FILE
BNE.S  ERROR   ;ERROR

MOVEQ.L  #100,DO  ;100 SECTORS ALLOCATED
LEA.L  FN(PC),A1  ;GET FILE NAME
XDFL   ;DEFINE CONTIGUOUS
BNE.S  ERROR

FN   DC.B  'FILENAME:EXT',0
EVEN
```
XDLF
Delete File

Value: $A0D6
Module: MPDOSF
Syntax: 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.

Possible Errors:
50 = Bad File Name
53 = File Not Defined
58 = File Delete or Write Protected
61 = File Already Open
68 = Not PDOS Disk
Disk errors

Example:

```
LEA.L FN(PC),A1 ;GET FILE NAME PTR
XDLF ;DELETE FILE
        ;ERROR
        ;NORMAL RETURN
        ....
FN DC.B 'TEMP/2',0
EVEN
```
XDMP

Dump Memory From Stack

Value: $A04A

Module: MPDOSK3

Syntax: 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:
XBUG - Debug Call
XRDM - Dump Registers
PB - PDOS Debugger (PDOS Monitor, Editor, Utilities manual)

Possible Errors:
None

Example:

START PEA.L START(PC)
MOVE.W #32,-(A7)
XDMP
XEXT
END START

x>MASM20 TEMP:SR, #TEMP

0000DDD0: 487A FFFE 3F3C 0020 A04A A00E 044F 5248 Hz.<.J...ORH
0000DDE10: 20CC 20C9 43EE 066E 4298 B1C9 65FA 2D49 .C...B...@.-1

PDOS ASSEMBLY PRIMITIVES REFERENCE  3.3 - 10/87
XDPE

Delay Physical Event

Value: $A114

Module: MPDOSK1

Syntax: XDPE

Registers:

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

Restrictions:

XDPE does not initialize the event like XDEV. You must initialize the event before using this call. If the event does not time out, clear it by setting the time to 0.

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:

- 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 680x0 bit number to use as an event (the 680x0 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:

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

Example:

```
... MOVE.L #$80800081,D1 ; SET DESCRIPTORS
LEA.L PEV(PC),A0 ; GET PEV ADDRESS
MOVEA.L A0,A1 ; COPY FOR EVI
MOVE.L #100,D0 ; SET TIMEOUT
BCLR.B D1,(A0) ; CLEAR TIMEOUT EV0
XDPE XSOE...
... PEV DC.W 0 ; SET VIA DELAY TIMER
```
XDTV

Define Trap Vectors

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XDTV</td>
</tr>
</tbody>
</table>
| Registers: | In D1.L = TVCZ FEDC BA98 7654 3210
(A0) = Table base address
(A1) = Vector table address
Vector table:
  DCL TRAP #0-<BASE ADR>
  ....
  DCL TRAP #15-<BASE ADR>
  DCL ZDIV-<BASE ADR>
  DCL CHK-<BASE ADR>
  DCL TRAPV-<BASE ADR>
  DCL TRACE-<BASE ADR> |

The vector table size is variable and each entry corresponds to non-zero bits in the mask register (D1.L). Each entry is a long signed displacement from the base address register.

The DEFINE TRAP VECTORS primitive loads user routine addresses into the task control block exception vector variables. Each task has the option to process its own TRAP, zero divide, CHK, TRAPV, and/or trace exceptions. Data register D1 selects which vectors are to be loaded according to individual bits corresponding to vectors in the vector table pointed to by address register A1. Bits 0 through 19 (right to left) correspond to TRAPs 0 through 15, zero divide, CHK, TRAPV, and trace exceptions. A 1 bit moves a vector from the vector table (biased by base address A0) into the task control block.

When an exception occurs, the task control block is checked for a corresponding non-zero exception vector. If found, then the return address is pushed on the user stack (USP) followed by the exception address and condition codes. PDOS next moves to user mode and executes a return with condition codes (RTR). This effectively acts like a jump subroutine with the return address on the user stack.
XDTV - Define Trap Vectors

IF <excp>$A6 THEN
1) Push return on USP
2) Push xxx$A6 on USP
3) Push CCs on USP
4) Move to user mode
5) Exit with RTR
ELSE PDOS error routine

The trace processing is handled differently. If the processor is in supervisor mode when a trace exception occurs, the trace bit is cleared and the exception is dismissed. The processor remains in supervisor mode. If the processor is in user mode and there is a non-zero trace variable in the task control block, then the trace is again disabled, the trace processor address is pushed on the supervisor stack along with status, and a return from exception is executed (RTE).

IF <sup> THEN
1) Disable trace
2) Exit in supervisor mode
ELSE IF TRCS$A6 THEN
1) Disable trace
2) Leave on stack
3) Push TRCS$A6
4) Push SR+$2000
5) Exit with RTE
ELSE PDOS error routine

Possible Errors:
None

See Also:
XVEC - Set/Read Exception Vector

Example:

<table>
<thead>
<tr>
<th>*</th>
<th>TVCZ/FEDCBA9876543210</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCON</td>
<td>EQU $11111000000000010001</td>
</tr>
<tr>
<td>SVECT</td>
<td>MOVE.L #VCON,D1 ;GET CONTROL VAR</td>
</tr>
<tr>
<td></td>
<td>LEA.L VT (PC),A0 ;POINT TO TABLE</td>
</tr>
<tr>
<td></td>
<td>MOVEA.L A0,A1 ;BASE=TABLE</td>
</tr>
<tr>
<td></td>
<td>XDTV ;SET VECTORS</td>
</tr>
<tr>
<td>VT</td>
<td>DC.L TRAP00-VT ;TRAP #0</td>
</tr>
<tr>
<td>DC.L</td>
<td>TRAP05-VT ;TRAP #5</td>
</tr>
<tr>
<td>DC.L</td>
<td>TRAP15-VT ;TRAP #15</td>
</tr>
<tr>
<td>DC.L</td>
<td>ZDIV-VT ;ZERO DIVIDE</td>
</tr>
<tr>
<td>DC.L</td>
<td>CHKP-VT ;CHK PROCESSOR</td>
</tr>
<tr>
<td>DC.L</td>
<td>TRPV-VT ;TRAPV PROCESSOR</td>
</tr>
<tr>
<td>DC.L</td>
<td>TRCE-VT ;TRACE</td>
</tr>
</tbody>
</table>
### XERR

**Return Error D0 to Monitor**

| **Value:** | $A00C |
| **Module:** | MPDOSK1 |
| **Syntax:** | XERR |
| **Registers:** | In \( \text{D0.W} = \text{Error code} \) |

**Description:**
The RETURN ERROR D0 TO MONITOR primitive exits to the PDOS monitor and passes an error code in data register D0. PDOS prints "PDOS ERR", followed by the decimal error number. The error call can be intercepted by changing the value of the ERR$ variable in the task TCB. This allows you to customize your own monitor.

**See Also:**
- XEXT - Exit To Monitor
- XEXZ - Exit To Monitor With Command

**Possible Errors:**
None

**Example:**

```assembly
XRSE ;READ SECTOR
BNE.S RERR ;ERROR
...

RERR CMPI.W #56,D0 ;EOF?
BNE.S RERR2 ;N
XCLF ;Y, CLOSE FILE
BNE.S RERR2
RTS
*
RERR2 XERR ;RETURN ERROR
```
## XEXC

**Execute PDOS Call D7.W**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A030</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XEXC</td>
</tr>
<tr>
<td>Registers:</td>
<td>In     D7.W = Aline PDOS CALL</td>
</tr>
<tr>
<td>Description:</td>
<td>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.</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>Call dependent</td>
</tr>
</tbody>
</table>
Example:

```assembly
****************************
* APPEND FILE
* AF <file1>,<file2>
* APDF MOVE.W #XAPF$,D7 ;APPEND COMMAND
  BRA.S RNFL02
*
****************************
* COPY FILE
* CF <file1>,<file2>
* CPYF MOVE.W #XCPY$,D7 ;COPY COMMAND
  BRA.S RNFL02
*
****************************
* RENAME FILE
* RN <file1>,<file2>
* RNFL MOVE.W #XRNF$,D7 ;RENAME COMMAND
  RNFL02 XGNP ;SOURCE FILE
    BLE.S ERR67
    MOVEA.L A1,A2 ;SAVE
    XGNP ;DESTINATION FILE
    BLE.S ERR67
    EXG.L A1,A2
    XEXC ;EXECUTE D7.W
    BNE.S RNFL04 ;ERROR
    XEXT ;RETURN
*
ERR67 MOVEQ.L #67,D0 ;PARAMETER ERROR
*
RNFL04 XERR ;ERROR
```
XEXT

Exit to Monitor

Value: $A00E
Module: MPDOSK1
Syntax: XEXT
(Always exits to monitor)
Registers: None
Description: The EXIT TO MONITOR primitive exits a user program and returns to the PDOS monitor. The exit can be intercepted by changing the value of the EXT$ variable in the task TCB. This primitive allows you to customize your own monitor.
See Also: XERR - Return Error D0 To Monitor
XEXZ - Exit To Monitor With Command
Possible Errors: None
Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCLF</td>
<td>;CLOSE FILE, ERROR?</td>
</tr>
<tr>
<td>BNE.S ERROR</td>
<td>;Y, DO ERROR CALL</td>
</tr>
<tr>
<td>XEXT</td>
<td>;N, RETURN TO MONITOR</td>
</tr>
<tr>
<td>Value:</td>
<td>$A04C</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XEXZ</td>
</tr>
<tr>
<td></td>
<td>(exits to monitor)</td>
</tr>
<tr>
<td>Registers:</td>
<td>In (A1) = Command string</td>
</tr>
<tr>
<td>Description:</td>
<td>The EXIT TO MONITOR WITH COMMAND primitive exits a user program and returns to the PDOS monitor. In addition, the monitor command buffer is loaded with the string pointed to by address register A1. This is useful in passing back parameters to the monitor or to chain to another program. The exit can be intercepted by changing the value of the EXT$ variable in the task TCB. This primitive allows you to customize your own monitor.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XERR - Return Error D0 To Monitor</td>
</tr>
<tr>
<td></td>
<td>XEXT - Exit To Monitor</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>None</td>
</tr>
<tr>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXIT LEA.L CMD(PC),A1 ;GET COMMAND</td>
</tr>
<tr>
<td></td>
<td>XEXZ ;EXIT</td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>CMD DC.B 'PRGM2',0</td>
</tr>
</tbody>
</table>
**XFAC**

**File Altered Check**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A0CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSF</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XFAC</td>
</tr>
<tr>
<td></td>
<td>&lt;status error return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>In (A1) = FILE NAME</td>
</tr>
<tr>
<td></td>
<td>Out CC = File not altered</td>
</tr>
<tr>
<td></td>
<td>CS = File altered</td>
</tr>
<tr>
<td></td>
<td>NE = Error</td>
</tr>
<tr>
<td>Description:</td>
<td>The FILE ALTERED CHECK primitive looks at the alter bit (bit $80) of the file pointed to by address register A1. If the bit is zero (not altered), then the primitive returns with the carry status bit clear. If the alter bit is set (file altered), then it is cleared and the primitive returns with carry set. If either case, the bit is always cleared.</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>Disk errors</td>
</tr>
<tr>
<td>Example:</td>
<td>XGNP ;GET PARAMETER</td>
</tr>
<tr>
<td></td>
<td>XFAC ;CHECK FOR FILE ALTERED</td>
</tr>
<tr>
<td></td>
<td>BNE.S @0002 ;ERROR</td>
</tr>
<tr>
<td></td>
<td>BCC.S FALSE ;NOT ALTERED, RETURN FALSE</td>
</tr>
<tr>
<td></td>
<td>BRA.S TRUE ;ALTERED, TRUE</td>
</tr>
</tbody>
</table>
**XFBF**

*Flush Buffers*

<table>
<thead>
<tr>
<th><strong>Value:</strong></th>
<th>$A0F8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module:</strong></td>
<td>MPDOSF</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td>XFBF</td>
</tr>
<tr>
<td></td>
<td>&lt;status error return&gt;</td>
</tr>
<tr>
<td><strong>Registers:</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Possible Errors:</strong></td>
<td>Disk errors</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>LOOP</td>
</tr>
<tr>
<td></td>
<td>MOVE.W #128, D1 ; EVEN 128</td>
</tr>
<tr>
<td></td>
<td>XDEV</td>
</tr>
<tr>
<td></td>
<td>XSUI</td>
</tr>
<tr>
<td></td>
<td>;SUSPEND</td>
</tr>
<tr>
<td></td>
<td>XFBF</td>
</tr>
<tr>
<td></td>
<td>;CHECK POINT DISK</td>
</tr>
<tr>
<td></td>
<td>BRA.S LOOP</td>
</tr>
</tbody>
</table>
XFFN

Fix File Name

<table>
<thead>
<tr>
<th>Value:</th>
<th>Value: $A0A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSF</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XFFN</td>
</tr>
<tr>
<td></td>
<td>&lt;status error return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>In (A1) = File name</td>
</tr>
<tr>
<td></td>
<td>Out D0,L = Disks(4th/3rd/2nd/1st)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TW0$ = Disk</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(A1) = MWB$, Fixed file name</td>
</tr>
<tr>
<td>Description:</td>
<td>The FIX FILE NAME primitive parses and verifies a character string for file name, extension, directory level, and disk number. The results are returned in the 32-character monitor work buffer (MWB$(A6)). Data register D0 is also returned with the disk numbers in the disk path. The first disk number in the disk path is returned in the monitor word temp (TW0(A6)). The error return is used for an invalid file name.</td>
</tr>
<tr>
<td></td>
<td>The filename convention is as follows where upper and lower case are unique:</td>
</tr>
<tr>
<td></td>
<td>APPPPPPP:PPP;NNN/NNN</td>
</tr>
<tr>
<td></td>
<td># -- Auto-create flag may prefix filename</td>
</tr>
<tr>
<td></td>
<td>A -- Alpha characters A-Z or a-z</td>
</tr>
<tr>
<td></td>
<td>P -- Printable characters except &quot;:&quot;, &quot;;&quot;, &quot;/&quot;. The &quot;.&quot; character may be used, but will conflict with the monitor command separator unless the filename is enclosed within parentheses</td>
</tr>
<tr>
<td></td>
<td>N -- Number in the range of 0-255</td>
</tr>
<tr>
<td></td>
<td>The monitor work buffer is cleared and the following assignments are made:</td>
</tr>
<tr>
<td></td>
<td>0(A1) = File name</td>
</tr>
<tr>
<td></td>
<td>8(A1) = File extension</td>
</tr>
<tr>
<td></td>
<td>11(A1) = File directory level</td>
</tr>
<tr>
<td></td>
<td>System defaults are used for the disk number and file directory level when they are not specified in the file name.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XRDN - Read Directory Entry By Name</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>50 = Bad File Name</td>
</tr>
<tr>
<td>Example:</td>
<td>XGLU ;GET INPUT LINE</td>
</tr>
<tr>
<td></td>
<td>XFFN ;FIX FILE NAME</td>
</tr>
<tr>
<td></td>
<td>BNE.5 ERROR ;ERROR IN NAME</td>
</tr>
</tbody>
</table>

52
### XFTD

**Fix Time and Date**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A058</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK3</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XFTD</td>
</tr>
</tbody>
</table>
| 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: | XPAD - Pack ASCII Date  
XRDT - Read Date  
XRTM - Read Time  
XUAD - Unpack ASCII Date  
XUDT - Unpack Date  
XUTM - Unpack Time |
| Possible Errors: | None |
| Example: | LEA.L TSTP(PC),A0 ;SAVE AREA  
XFTD ;GET TIME STAMP  
MOVEM.W D0-D1,(A0) ;SAVE TIME & DATE  
....  
TSTP DS.W 2 ;TIME STAMP SAVE |
XFUM

Free User Memory

Value: $A040

Module: MPDOSKI

Syntax: 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.

The number passed to D0.W must be an even number since memory that is allocated or deallocated must be in 2K increments. If the number is odd, it will be rounded up to a 2K boundary. If D0=0, no action is taken. If D0<0 then error 79 will occur.

Possible Errors:

79 = Bad Memory Address

Example:

```
MOVEQ.L $20,D0 ;FREE 20K
MOVEA.L A2,A0 ;AT A2
XFUM ;FREE MEMORY
BNE.S ERROR
```
Conditioned Get Character

Value: $A048
Module: MPDOSK2
Syntax: XGCB
<status return>

Registers:
- **Out**
  - D.O.L = Character in bits 0-7
- **SR**
  - EQ....No character
  - LO....Ctrl C
  - LT....Esc
  - MI....Ctrl C or Esc

Description:
The CONDITIONAL GET CHARACTER primitive checks for a character from first, the input message pointer (IMPS(A6)), second, the assigned input file (ACIS(A6)), and then finally, the interrupt driven input character buffer (PRTS(A6)). If a character is found, it is returned in the right byte of data register D.O.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

Example:

```
LOOP XGCB ;CHARACTER?
BEQ.S NONE ;N
BLO.S QUIT ;Y, ^C, DONE
BLT.S NEXT ;CONTINUE
CMPI.B #'0',DO ;NUMBER?
...`
```
Get Character Conditional

Value: $A078
Module: MPDOSK2
Syntax: 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

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

Example:

```
  ......  ;CHARACTER?
   XGCC  ;Y, WAIT CHARACTER
    BEQ.S CONT  ;N, CONTINUE
    BLO.S QUIT  ;Y, ^C, QUIT
    BLT.S NEXT  ;Y, ESC, GOTO NEXT
  *   WAIT   XGCR  ;Y, WAIT CHARACTER
  *  CONT  .....
```
XGCP
Get Port Character

Value: $A09E
Module: MPDOSK2
Syntax: XGCP
<status return>

Registers:
Out  D0.L = Character in bits 0-7
     SR =  LO...Ctrl C
      LT...Esc
      MI...Ctrl C or Esc

Description:
If the ignore control character bit ($02) of the port flag is set, then XGCP ignores Ctrl C and Esc.

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:
86 = Suspend on Port 0

Example:

```
LOOP XGCP ;GET PORT CHARACTER
  BLO.S QUIT ;C, DONE
  BLT.S NEXT ;CONTINUE
  CMPI.B #’O’, D0 ;NUMBER?
  ....
```

PDOS ASSEMBLY PRIMITIVES REFERENCE 3.3 - 10/87 57
XGCR
Get Character

Value: $A07A
Module: MPDOSK2
Syntax: XGCR
<status return>
Registers:
Out
D0.L = Character in bits 0-7
SR = LO....Ctrl C
LT....Esc
MI....Ctrl C or Esc

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. However, if the "receiver interrupt disable" bit is set on the port, the UART type is polled for a character. If there is a character from the UART, then it is placed in the type ahead buffer.

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:
86 = Suspend on Port 0

Example:

```
LOOP XGCR ;GET CHARACTER
BLO.QUIT ;"C", DONE
BLT.S NEXT ;CONTINUE
CMPI.B #'0',D0 ;NUMBER?
.....
```
XGLB
Get Line in Buffer

Value: $A07C
Module: MPDOSK2
Syntax: XGLB
          {BLT.x ESCAPE}  optional
          <status return>

Registers:
In  (A1)  = Buffer address
Out D1.L  = Number of characters
      SR =    EQ,...,J only
      LT...Esc
      LO...Ctrl C

Description:

If the ignore control character bit ($02) of the port flag is set, then XGLB ignores Ctrl C and Esc.

The GET LINE IN BUFFER primitive gets a character line into the buffer pointed to by address register A1. The XGCR primitive is used by XGLB and hence characters can come from a memory message, a file, or the task console port.

The buffer must be at least 80 characters in length. The line is delimited by a carriage return. The status returns EQUAL if only a .J is entered.

If an Esc is entered, the task exits to the PDOS monitor unless a "BLT" instruction immediately follows the XGLB call. If such is the case, then XGLB returns with status set at "LT".

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

The command line can be edited with various system defined control characters. A Backspace ($08) moves the cursor one character to the left. A Ctrl F ($0C) moves the cursor one character to the right. A Del ($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 .J is entered.

See Also: XGLU - Get Line In User Buffer

Possible Errors: None
XGLB - Get Line in Buffer

Example:

```
OPEN XPMC MES01 ;PROMPT
LEA.L BUF(PC),A2 ;GET BUFFER ADDRESS
XGLB ;GET LINE IN BUFFER
BLT.S OPEN ;DO NOT EXIT ON ESC
BEQ.S OPEN10 ;USE DEFAULT
*
OPEN2 XSOP ;OPEN FILE
BNE.S OPEN4 ;ERROR
....
OPEN4 CMPI.W $53,DO ;'NOT DEFINED' ERROR?
BNE.S OPERR ;N
XDFL ;Y, DEFINE FILE, ERROR?
BEQ.S OPEN2 ;N
*
OPERR XERR ;Y, REPORT ERROR
*
OPEN10 ....

MES01 DC.B $GA,$OD,'FILE=',0
BUF DS.B 80
```
**XGLM**

Get Line in Monitor Buffer

| **Value:** | $A07E |
| **Module:** | MPDOSSK2 |
| **Syntax:** | XGLM 
  \( (BLT.x \text{ ESCAPE}) \) _optional_ 
  \(<\text{status return}>\) |
| **Registers:** | Out \( (A1) = \text{String} \) 
  D1.L = Number of characters 
  SR = \( \text{EQ} \ldots \text{J only} \) 
  LT...Esc 
  LO...Ctrl C |

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 \( \text{J} \) is entered.

If an Esc is entered, the task exits to the PDOS monitor unless a “BLT” instruction immediately follows the XGLM call. If such is the case, then XGLM returns with status set at “LT”.

If the assigned console flag (ACIS(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 Del ($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 \( \text{J} \) 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

**Example:**

```
XGLM ;GET LINE
BEQ.S NONE
```
XGLU

Get Line in User Buffer

Value: $A080
Module: MPDOSK2
Syntax: XGLU
          {BLT.x ESCAPE}   optional
          <status return>

Registers:
Out (A1) = String
       D1.L = Number of characters
       SR =  EQ...j only
              LT...Esc
              LO...Ctrl C

Description:
If the ignore control character bit ($02) of the port flag is set, then XGLU ignores Ctrl C and Esc.

The GET LINE IN USER BUFFER primitive gets a character line into the user buffer. Address register A6 normally points to the user buffer. The XGCR primitive is used by XGLU; hence, characters come from a memory message, a file, or the task console port. The line is delimited by a carriage return. The status returns EQUAL if only a .J is entered. Address register A1 is returned with a pointer to the first character.

The user buffer is located at the beginning of the task control block and is 256 characters in length. However, the XGLU routine limits the number of input characters to 78 plus two nulls.

If an Esc ($1B) is entered, the task exits to the PDOS monitor unless a “BLT” instruction immediately follows the XGLU call. If such is the case, then XGLU returns with status set at “LT”.

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

The command line can be edited with various system defined control characters. A Backspace ($08) moves the cursor one character to the left. A Ctrl L ($0C) moves the cursor one character to the right. A Del ($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 .J is entered.

Possible Errors:
None
Example:

```
GETN    MOVEQ.L #DNUM,D4    ;GET DEFAULT #
XGLU    ;GET LINE
BEQ.S  GETN2    ;USE DEFAULT
XCBD    ;CONVERT #, ERROR?
BLE.S  ERROR    ;Y
    MOVE.L  D1,D4    ;N

* GETN2    MOVE.L  D4,-(A7)    ;SAVE #
```
### Get Memory Limits

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSS1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XGML</td>
</tr>
<tr>
<td>Registers:</td>
<td></td>
</tr>
</tbody>
</table>

- **Out**
  - (A0) = End TCB (TBES)
  - (A1) = Upper memory limit (EUMS-USZ)
  - (A2) = Last loaded address (BUMS)
  - (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

**Example:**

```assembly
START XGML ;GET MEMORY LIMITS
*
START2 CLR.B (A2)+ ;CLEAR MEMORY
CMPA.L A1,A2 ;DONE?
BLO.S START2 ;N
```
**XGMP**

**Get Message Pointer**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A004</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
</tbody>
</table>
| Syntax: | XGMP  
<status return> |
| Registers: | In  
D0.B = Message slot number (0..15)  
D0.L = Source task # (-1 = no message)  
SR =  
   EQ....Message (Event[64+Message slot #]=0)  
   NE....No message  
   D0.L = Error number 62 if message pointer error  
(A1) = Message  
Out |
| Description: | The GET MESSAGE POINTER primitive looks for a task message pointer. If no message is ready, then data register D0 returns the error number 62 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: | XGTM - Get Task Message  
XKTM - Kill Task Message  
XSMP - Send Message Pointer  
XSTM - Send Task Message |
| Possible Errors: | 62 = Bad Message Ptr Call |
| Example: | . . .  
MOVE.W #69,D1  
XSUI  
MOVE.B #D,D0  ;Check message slot #5  
XGMP  
BNES.S NOMESS  ;No message  
XPMC  ;Print message to console  
. . .  
NOMESS  
XPMC  
MESS  
. . .  
MESS  
DC.B  $0A,$0D,‘NO MESSAGE POINTER’,0 |
The GET NEXT PARAMETER primitive parses the monitor buffer for the next command parameter. The routine does this by maintaining a current pointer into the command line buffer (CLB$) and a parameter delimiter (CMD$).

The XGNP primitive clears all leading spaces of a parameter. A parameter is a character string delimited by a space, comma, period, or null. If a parameter begins with a left parenthesis, then all parsing stops until a matching right parenthesis or null is found. Hence, spaces, commas, and periods are passed in a parameter when enclosed in parentheses. Parentheses may be nested to any depth.

An "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.

None
Example:

```
SPAC MOVE.B SDK$ (A6), D0 ; GET SYSTEM DISK #
XGNP ; GET PARAMETER, OK?
BLS. S SPAC02 ; N, USE DEFAULT
XCDB ; Y, CONVERT, OK?
BLE.S ERR67 ; N, ERROR
MOVE.L D1, D0 ; Y
*
SPAC02 XSZF ; GET DISK SIZE
BNE.S ERROR ; PROBLEM
....
```
XGTM
Get Task Message

Value: $A01E
Module: MPDOSK1
Syntax: 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:
XGMP - Get Message Pointer
XKTM - Kill Task Message
XSMP - Send Message Pointer
XSTM - Send Task Message

Possible Errors: None

Example:

```
LOOP    LEA.L  BUF(PC),A1  ;GET BUFFER ADR
        XGTM    ;LOOK FOR MESSAGE
        BNE.S  NONE   ;NONE
        XPCL    ;OK, OUT CRLF
        XPLC    ;OUT MESSAGE
        BRA.S LOOP  ;LOOK AGAIN
*
    NONE    ....
    BUFFER DS.B 64 ;MESSAGE BUFFER
```
### XGUM

**Get User Memory**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A03E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
</tbody>
</table>
| Syntax: | XGUM  
<status error return> |
| Registers: | In  
DO.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 DO.W Kbytes. If found, the "EQ" status is set, address registers A0 and A1 are returned the the start and end memory address, and the memory block is marked as allocated in the bit map.  
The number in register DO must be an even number. Memory is both allocated and deallocated in 2K blocks. |
| See Also: | XFUM - Free User Memory |
| Possible Errors: | 73 = Not Enough Memory |
| Example: | ```
GETM CLR.W -(A7) ;PUSH .NE.
MOVEQ.L #10,D0 ;GET 10K BYTES
XGUM
BNE.S @GM02 ;ERROR
MOVE.L A0,AV(A6) ;SAVE
ADQ.W #$04,(A7) ;RETURN .EQ.
*
@GM02 RTR ;RETURN
``` |
**XISE**

Initialize Sector

**Value:** $A0C0

**Module:** MPDOSF

**Syntax:**

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:**

- BIOS in *PDOS Developer's Reference Manual*
- XRSE - Read Sector
- XRSZ - Read Sector Zero
- XWSE - Write Sector

**Possible Errors:**

- Disk errors

**Example:**

```assembly
MOVEQ.L DSKN, D0 ;GET DISK #
MOVEQ.L #0, D1 ;START AT SECTOR 0
LEA.L BUF(PC), A2 ;GET BUFFER PTR

* LOOP:
  XISE ;WRITE TO DISK
  BNE.S ERROR ;ERROR
  ADDQ.W #1, D1 ;MOVE TO NEXT
  CMPI.W #DISKZ, D1 ;DONE?
  BLO.S LOOP ;N

....
```
**XKTB**

**Kill Task**

**Value:** $A0FA

**Module:** MPDOSK1

**Syntax:**

```
XKTB
<status error return>
```

**Registers:**

In  

D0.B = Task number

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.

If D0=0, then kill self & deallocate memory

The task number is specified in data register D0.B. If register D0.B equals zero, then the current task is killed and its memory deallocated in the system memory bit map.

If D0>0, then kill task D0 & deallocate memory

If D0<0, then kill task ABS(D0) & do not deallocate memory

If D0.B is positive, then the selected task is killed and its memory deallocated. If D0.B is negative, then task number ABS(D0.B) is killed, but its memory is not deallocated in the memory bit map.

**See Also:**

XCTB - Create Task Block

**Possible Errors:**

74 = Non-existent Task

**Example:**

```
PREND CLR.B D0 ;KILL SELF
XKTB ;CALL CURRENT TASK
BNE.S ERROR
```
**XKTM**

**Kill Task Message**

| **Value:** | $A028 |
| **Module:** | MPDOSK1 |
| **Syntax:** | 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:** | XGMP - Get Message Pointer  
XGTM - Get Task Message  
XSMP - Send Message Pointer  
XSTM - Send Task Message |
| **Possible Errors:** | None |
| **Example:** |  
```assembly  
LOOP  MOVEQ.L #0,D0 ;SELECT TASK 0  
LEA.L BF(PC),A1  
XKTM ;ANY MESSAGE?  
BEQ.S LOOP ;Y, DO AGAIN  ``` |
Value: $A0B0
Module: MPDOSF
Syntax: 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 or “OB” entry address
(A1) = BUM$ - Last loaded address

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 or binary 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. Any type of file may be loaded if the execution flag is clear. If D1.B<>0, then 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.

For “OB” type files, section 0 code is loaded first followed by section 1 and so forth to section 15. All simple references among sections are resolved but no operations are allowed. The loader also sets the task entry address EAD$(A5) and register A0 to the address specified by the start tag, or to the start of the file if no start address is given. All object files must be assembled with the 3.3 assembler in order to load.

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.
**XLDF - Load File**

Legal tags:
- 0t--LABEL--vvvrrdddddttttt
- 1Saaaaaaa ;ENTRY POINT
- 2Saaaaaaa ;ADDRESS
- 3dd ;SIMPLE DATA BYTE
- 4dddd ;SIMPLE DATA WORD
- 5ddddddddd ;SIMPLE LONG DATA WORD
- 6 ;POP BYTE
- 7 ;POP WORD
- 8 ;POP LONG WORD
- 9Snnnnnnnn ;PUSH VALUE
- Dcccddd ;STORE MULTIPLE WORD
- ES11111111 ;SECTION LENGTH
- Fcc ;END OF RECORD/CHECKSUM

Illegal tags:
- Asl<symbol> ;PUSH SYMBOL
- BO ;DO OPERATION
- CS1<symbol>nnnnnnnn ;EXTERNAL DEFINITION

Possible Errors:
63 = Bad Object Tag
65 = Not Executable
73 = Not Enough Memory
Disk errors

Example:

```
XGML ;GET MEMORY LIMITS
CLR.L DO ;RETURN
ADDA.W #$100,A0 ;ADD DISPLACEMENT
LEA.L FN(PC),A3 ;GET FILE NAME
XLDF ;LOAD FILE
BNE.S ERROR ;ERROR
```
**Value:** $A03A

**Module:** MPDOSK1

**Syntax:** 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 "&O" during a procedure file.

User programs should execute this call when an error occurs.

The enable echo flag (ECFS(A6)) is cleared by this call.

**Possible Errors:** None

**Example:**

```
ADDI.W #300,DO ;BIAS ERROR #
XLER ;REPORT TO PDOS
```
XLFN

Look for Name in File Slots

Value: $A0A2

Module: MPDOSF

Syntax: 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

If D3.W=0, then no slots are available.

The LOOK FOR NAME IN FILE SLOTS primitive searches through the file slot table for the file name as specified by registers D0.B and A1. If the name is not found, register D3.W returns with a -1 or 0. The latter indicates the file was not found and there are no more slots available. Otherwise, register D3.W returns the associated file ID and register A3 returns the address of the file slot.

A file slot is a 38-byte buffer where the status of an open file is maintained. There are 32 file slots available. The file ID consists of the disk # and the file slot index.

File slots assigned to read-only files are skipped and not considered for file match.

File slot format: (38 bytes)
   0(A3) = File name.11
   11(A3) = Level.1
   12(A3) = Status.2
   14(A3) = Sector # in memory.2
   16(A3) = Pointer.4
   20(A3) = Sector index in memory.2
   22(A3) = Sector index of eof.2
   24(A3) = # bytes in end sector.2
   26(A3) = Lock.1/shared flag.1
   28(A3) = Channel buffer ptr.4
   32(A3) = Lock.1/shared flag.1
   34(A3) = Roll-out error #.2
   36(A3) = Disk #.2
XLFN - Look for Name in File Slots

Possible Errors:

None

Example:

XNOP LEA.L FN(PC),A1 ;POINT TO FILE NAME
XPFN ;FIX FILE NAME
BNE.S ERR1 ;ERROR
XLFN ;LOOKUP NAME, FOUND?
BEQ.S ERR2 ;Y, FILE ALREADY OPEN
....

ERR1 XPMC MERR1 ;INVALID FILE NAME
RTS
*
ERR2 XPMC MERR2 ;FILE ALREADY OPEN
RTS
*
FN DC.B 'FILENAME',0
MERR1 DC.B $0A,$0D,'INVALID FILE NAME',0
MERR2 DC.B $0A,$0D,'FILE ALREADY OPEN',0
EVEN

Status: $01xx Sequential
$02xx Random
$06xx Shared random
$0Axx Read only random
$10xx Driver in channel
....
$xx00 Altered
$xx04 Contiguous
$xx02 Delete protect
$xx01 Write protect
$8xxx Sector altered
$4xxx File altered
$2xxx Buffer locked in memory
**XLKF**

**Lock File**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A0D8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSF</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XLKF &lt;status error return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>In D1.W = File ID</td>
</tr>
<tr>
<td>Description:</td>
<td>The LOCK FILE primitive locks an opened file so that no other task can gain access until an UNLOCK FILE (XULF) primitive is executed. Only the locking task has access to the locked file. A locked file is indicated by a -1 ($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).</td>
</tr>
<tr>
<td>See Also:</td>
<td>XULF - Unlock File</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>52 = File Not Open 59 = Bad File Slot 75 = File Locked Disk errors</td>
</tr>
<tr>
<td>Example:</td>
<td>MOVE.W D5,D1 ;GET FILE ID XLKF ;LOCK FILE BNE.S ERROR ;PROBLEM ....</td>
</tr>
</tbody>
</table>
XLKT
Lock Task

Value: $A014
Module: MPDOSK1
Syntax: 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:
XULT - UNLOCK TASK

Possible Errors:
None

Example:

```
XLKT ;LOCK TASK
SNE.B D7 ;SET FLAG
TAS.B SBIT ;START CRITICAL PROCESS
*
WAIT TST.B SBIT ;OK?
BMI.S WAIT ;N
TST.B D7 ;Y, LEAVE LOCKED?
BNE.S CONT ;Y
*
XULT ;N, UNLOCK TASK
*
CONT ....
```
# XLSR

## Load Status Register

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A02E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XLSR</td>
</tr>
<tr>
<td>Registers:</td>
<td>D1.W = 68000 status register</td>
</tr>
</tbody>
</table>

### 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:

- XRSR - Read Status Register
- XSUP - Enter Supervisor Mode

### Possible Errors:

None

### Example:

```
MOVE.W SR,D1 ;READ STATUS
ORI.W #$2000,D1 ;ADD SUPERVISOR
XLSR ;LOAD SR
```
XLST

List File Directory

Value: $A0A4
Module: MPDOSM
Syntax: XLST
<status error return>

Registers: 

In (A1) = List specifications

Description:
The LIST FILE DIRECTORY subroutine causes PDOS to output a formatted file directory listing to the console terminal, according to the select string pointed to by address register A1. The output may be interrupted at any time by a character being entered on the console port. An Esc character returns control to the PDOS monitor.

The format of the list specifications is defined as follows:

```
DC.B '{file}{:ext}{;level}{/disk}{/select...}',0
```

where:

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

Possible Errors:

Example:

```
MLST XGNP ;GET SELECT LIST
XLST ;CALL FOR LIST
BNE.S ERROR ;ERROR
XXXT ;EXIT TO MONITOR
```
**XNOP**

**Open Shared Random File**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A0DA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSF</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XNOP</td>
</tr>
<tr>
<td></td>
<td>&lt;status error return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>In (A1) = File name</td>
</tr>
<tr>
<td></td>
<td>Out D0.W = File attribute</td>
</tr>
<tr>
<td></td>
<td>D1.W = File ID</td>
</tr>
</tbody>
</table>

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.

\[
D0.W = (ABOS \ BETU xxxx xCWD) \\
D1.W = (Disk \#) \times 256 + (file \ slot \ index)
\]

The END-OF-FILE marker on a shared file is changed only when the file has been extended. All data transfers are buffered through a channel buffer; data movement to and from the disk is by full sectors.

An "opened count" is incremented each time the file is shared-opened and is decremented by each close operation. The file is only closed by PDOS when the count is zero. This count is saved in the right byte of the locked file parameter (LF) and is listed by the file slot usage command (FS).

<table>
<thead>
<tr>
<th>Possible Errors:</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 = Bad File Name</td>
</tr>
<tr>
<td>53 = File Not Defined</td>
</tr>
<tr>
<td>60 = File Space Full</td>
</tr>
<tr>
<td>61 = File Already Open</td>
</tr>
<tr>
<td>68 = Not PDOS Disk</td>
</tr>
<tr>
<td>69 = Out of File Slots</td>
</tr>
<tr>
<td>Disk errors</td>
</tr>
</tbody>
</table>
Example:

```
LEA.L FN(PC),P1 ;POINT TO NAME
XNOP ;OPEN SHARED
BNE.S ERROR
MOVE.W D0,D5 ;SAVE TYPE
SWAP D5
MOVE.W D1,D5 ;SAVE FILE ID

FN DC.B 'FILENAME:EXT',0
EVEN
```
XPAD

Pack ASCII Date

Value: $A00A

Module: MPDOSK3

Syntax: XPAD

Registers:

In

(A1) = 'DY-MON-YR'

Out

D1.W = (Year*16+month)*32+day

(YYYY YYYY MMDD DDDD)

(A1) = Updated

SR = .EQ. - Conversion okay

.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:

XFTD - Fix Time And Date
XRDT - Read Date
XRTM - Read Time
XUAD - Unpack ASCII Date
XUDT - Unpack Date

Possible Errors:

Status errors

Example:

```
STRT XPMC MES1 ;DATE=
XGLU ;GET LINE
XPAD ;CONVERT
BNE.S ERR ;ERROR
XPMC MES2 ;D1.W=
XCBH
ADDQ.W #4,41
XPIC ;OUTPUT
BRA.S STRT
*
ERR XPMC MES3 ;ERROR
BRA.S STRT
*
MES1 DC.B $0A,$0D,'DATE=',0
MES2 DC.B ' D1.W=$',0
MES3 DC.B $0A,$0D,'*ERROR=',0
EVEN
END STRT
```

*x>TEST

DATE=11-NOV-86 D1.W=$AD6B
DATE=11NOV86 D1.W=$AD6B
DATE= NOV 11 86
*ERROR
DATE=-

84
Value: $A084
Module: MPDOSK2
Syntax: 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 \( ^\leftarrow \) ($0D) clears the counter. Tabs ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNTS$(A6)) and spool unit (SPUS$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPIS$(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:
XGLB - Get Line In Buffer

Possible Errors:
None

Example:
```
CLINE MOVEA.L A6,A2 ;GET USER BUFFER PTR *
    CLINE2 
    ........
    MOVE.B D0,(A2)+ ;LOAD BUFFER, DONE?
    BNE.S CLINE2 ;N
    XPBC ;Y, OUTPUT BUFFER
    RTS ;CONTINUE
```
**XPCB**

**Push Command to Buffer**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A04E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSM</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XPCB</td>
</tr>
<tr>
<td>Registers:</td>
<td>In ( (A1) = \text{Command string} )</td>
</tr>
<tr>
<td>Description:</td>
<td>The PUSH COMMAND TO BUFFER primitive pushes the string pointed to by address register A1 into the command recall buffer. Since there is a limit on the buffer size, older commands are lost.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XGNP - Get Next Parameter</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>None</td>
</tr>
<tr>
<td>Example:</td>
<td></td>
</tr>
</tbody>
</table>

```assembly
XGLU ;GET COMMAND
XPCB ;PUSH FOR RECALL
...```


### XPCC

#### Put Character(s) to Console

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A086</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XPCC</td>
</tr>
<tr>
<td>Registers:</td>
<td>In D0.W = Character(s)</td>
</tr>
<tr>
<td>Description:</td>
<td>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 $ (0D) clears the counter. Tabs ($09) are expanded with blanks to MOD 8 character zone fields. If there are coinciding bits in the unit (UNTS(A6)) and spool unit (SPUS(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(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.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XPCR - Put Character Raw</td>
</tr>
<tr>
<td></td>
<td>XPDC - Put Data To Console</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>None</td>
</tr>
</tbody>
</table>
| Example:     | MOVE.W #'C',D0 ;OUTPUT 'C'  
|              | XPCC  
|              | MOVEQ.L #$0A,D0 ;FOLLOWED BY LF  
|              | XPCC |

**Example:**

```
MOVE.W #'C',D0 ;OUTPUT 'C'
XPCC
MOVEQ.L #$0A,D0 ;FOLLOWED BY LF
XPCC
```
XPCL

Put CRLF to Console

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A088</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XPCL</td>
</tr>
<tr>
<td>Registers:</td>
<td>None</td>
</tr>
</tbody>
</table>
| Description: | The PUT CRLF TO CONSOLE primitive outputs the ASCII characters line feed <$0D> and carriage return <$0A> 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.

<table>
<thead>
<tr>
<th>Possible Errors:</th>
<th>None</th>
</tr>
</thead>
</table>

Example:

```
XPCL ;OUTPUT CRLF
...```


XPCP
Place Character in Port Buffer

Value: $AOBC
Module: MPDOSK2
Syntax: 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 PDOS port from a task or program.

Example:

```
START LEA.L STRING(PC),A0 ;ADDRESS OF STRING
   MOVE.W #3,D1  ;PLACE IN PORT 3 INPUT BUFFER
   *  LOOP MOVE.B (A0)+,DO ;GET CHAR, TEST FOR 0?
                 ,BEQ.S DONE ;Y
                 ,XPCP ;PUT INTO PORT 3 INPUT
   BRA.S LOOP
   *  DONE XEXT
   *  STRING DC.B 'HELLO PORT 3!',0
   *  EVEN
   END START

>MAST TEST: SR, TEST:J
>TEST:J
>TM 3,2HELLO PORT 3!Ctrl B.J
```

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 6 is cleared.

The port specified in the XPCP call is independent of windowing — it refers to the physical port, not the logical port.
### XPCR

**Put Character Raw**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A0BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XPCR</td>
</tr>
<tr>
<td>Registers:</td>
<td>In D0.B = CHARACTER</td>
</tr>
</tbody>
</table>

**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:**
XPCC - Put Character(s) To Console
XPDC - Put Data To Console

**Possible Errors:**
None
XPDC
Put Data to Console

Value: $A096
Module: MPDOSK2
Syntax: 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:
XPCC - Put Character(s) To Console
XPCR - Put Character Raw

Possible Errors:
None

Example:

MOVEQ.L #0,D7
LEA.L M(PC),A1 ;POINT TO STRING
MOVE.B (A1)+,D7 ;GET LENGTH
XPDC ;OUTPUT
...

M DC.B 10,50A,$0D
DC.B 'THIS IS A MESSAGE'
Put Encoded Line to Console

Value: $A06E
Module: MPDOSK2
Syntax: 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 J ($0D) clears the counter. Tabs ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNTS(A6)) and spool unit (SPUS(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPIS(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:
XPEM - Put Encoded Message To Console
XPLC - Put Line To Console
XPMC - Put Message To Console

Possible Errors:
None

Example:

LEA.L M(FC),A1 ;POINT TO MESSAGE
XPEL ;OUTPUT MESSAGE

M
DC.B $80,'Lev','-2','Name:ext'
DC.B -6,'Type','-6','Size','-6
DC.B 'Dat','-e','created','-4
DC.B 'Las','-t','update',0
XPEL - Put Encoded Line to Console

The above text strings are equivalent to:

```
M     DCE.B  $80,'Lev Name:ext'
DCE.B  '      Type     Size'
DCE.B  '      Date created'
DCE.B  '      Last update',0
```

($80 is equal to a CR/LF)
**XPEM**

Put Encoded Message to Console

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A09C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XPEM &lt;message&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>None</td>
</tr>
</tbody>
</table>
| Description: | The PUT ENCODED MESSAGE TO CONSOLE primitive outputs the PC relative message contained in the word following the call to the user console. 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 $ (S0D) clears the counter. Tabs ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPUS$(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:**
XPEL - Put Encoded Line To Console
XPLC - Put Line To Console
XPMC - Put Message To Console

**Possible Errors:**
None

**Example:**

```assembly
XPEM MES01 ;OUTPUT MESSAGE
.....

MES01 DC.B $80,'Lev','-2','Name:ext'
DC.B '-6','Type','-6','Size','-6
DC.B 'Dat','-8','created','-4
DC.B 'Las','-8','update',0

$80 = Carriage return/line feed
```
**XPLC**

**Put Line to Console**

**Value:**  
$A08A

**Module:**  
MPDOSK2

**Syntax:**  
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 .J ($0D) clears the counter. Tabs ($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:**

XPEL - Put Encoded Line To Console  
XPEM - Put Encoded Message To Console  
XPMC - Put Message To Console

**Possible Errors:**  
None

**Example:**

```
LEA.L MESS(PC),A1 ;OUTPUT MESSAGE
XPLC
MOVE.L NUMB(PC),D1 ;GET NUMBER
XCBD ;CONVERT TO DECIMAL
XPLC ;OUTPUT
....

NUMB DS.L 1 ;NUMBER HOLDER
MESS DC.B $0A,$0D ;MESSAGE #1
DC.B 'ANSWER=',0
```
The PUT MESSAGE TO CONSOLE primitive outputs the ASCII character string pointed to by the message address word immediately following the PDOS call to the user console and/or SPOOL file. The address is a PC relative 16-bit displacement to the message. The output string is delimited by the null character.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A Backspace ($08) decrements the counter while a .J ($0D) clears the counter. Tabs ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNTS(A6)) and spool unit (SPUS(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPIS(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.

**Example:**

```
XPMC       ;OUTPUT HEADER
          ****
MES2       DC.B    $0A,$0D ;HEADER MESSAGE
DC.B     'PDOS REV 3.0',0
```
### XPSC

**Position Cursor**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A08E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XPSC</td>
</tr>
</tbody>
</table>
| Registers: | In  
D1.B = Row  
D2.B = Column |

Uses PSC$(A6) as lead characters.

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.

```
PSCS(A6)= B111 1111 0222 2222
         B = 0 then $00 bias; =1 then $20 bias
         0 = 0 send row first then column
             1 send column then row
         1 = 7 bits for first ASCII lead in character
         2 = 7 bits for second ASCII lead in character
```

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.

If the high bit is set in the PSCS.W then the row and column characters are biased by $20; otherwise, they have a zero bias. If the parity bit in the low order byte is zero, then the sending order is row/column; otherwise, it is reversed.

If PSC$ is zero or if the first byte equals $FF, then PDOS makes a call into the BIOS for custom position cursor with a $20 bias. The entry point is B_PSC beyond the BIOS table. If the high order byte of PSC$ is -1, PDOS makes a call into the BIOS at B_PSC beyond byte of PSC$ and executes the proper code depending on the value found in the low order byte.

The MTERM utility is used to change the position cursor codes. MTERM will not handle calls to the BIOS for custom position cursor.

**See Also:**
- XCLS - Clear Screen
- XRCP - Read Port Cursor Position
- BIOS in *PDOS Developer's Reference Manual*
XPSC - Position Cursor

Possible Errors:

Example:

```
OUTM MOVEQ.L #23,D1 ;POSITION TO BOTTOM
CLR.L D2 ; OF SCREEN
XPSC ;POSITION
XPMC MES1 ;OUTPUT MESSAGE
....
```
XPSF

Position File

Value: $A0DC

Module: MPDOSF

Syntax:

XPSF  
<status error return>

Registers:

In   D1.W = File ID  
     D2.L = Byte position

Description:

A byte position equal to -1 positions to the end of the file.

The POSITION FILE primitive moves the file byte pointer to any byte position within a file. The file ID is given in register D1 and the long word byte position is specified in register D2.

An error occurs if the byte position is greater than the current end-of-file marker.

A contiguous file greatly enhances the speed of the position primitive since the desired sector is directly computed. However, the position primitive does work with non-contiguous files, as PDOS follows the sector links to the desired byte position.

A contiguous file is extended by positioning to the end-of-file marker and writing data. However, PDOS will alter the file type to non-contiguous if a contiguous sector is not available. This would result in random access being much slower.

See Also:

XRFP - Read File Position  
XRWF - Rewind File

Possible Errors:

52 = File Not Open  
59 = Bad File Slot  
70 = Position EOF  
Disk errors

Example:

```
MOVE.W D5,D1  ;GET FILE ID  
MOVE.W RN(A0),D2  ;GET RECORD #  
MULU.W #36,D2  ;GET BYTE INDEX  
XPSF  ;POSITION WITHIN FILE  
BNE.S ERROR  
odds  
RN  
D5.W 1  ;RECORD #
```
**XPSP**

**Put Space to Console**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A098</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK2</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XPSP</td>
</tr>
<tr>
<td>Registers:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>The PUT SPACE TO CONSOLE outputs a Space ($20) character to the user console. There are no registers or status involved. If there are coinciding bits in the unit (UNT$(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.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XPCC - Put Character(s) To Console</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>None</td>
</tr>
</tbody>
</table>
| Example:               | MOVEQ.L #N,D1 ;GET NUMBER  
                         | XCBM MES01 ;CONVERT  
                         | XPLC ;OUTPUT LINE  
                         | XPSP ;OUT SPACE |


XRBF

Read Bytes From File

Value: $A0DE

Module: MPDOSF

Syntax: 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: XRLF - Read Line From File
XWBF - Write Bytes To File
XWLF - Write Line To File

Possible Errors: 52 = File Not Open
56 = End Of File
59 = Bad File Slot
Disk errors

Example:

MOVE.L #256, D0 ;READ 256 BYTES
MOVE.W D5, D1 ;GET FILE ID
MOVEA.L A6, A2 ;READ INTO USER BUF
XRBF ;READ DATA
BNE.S ERROR

ERROR CMP.L #56, D0 ;EOF?
BNE.S ERROR2 ;N
MOVE.L D3, D0 ;Y, GET # OF BYTES READ

PDOS ASSEMBLY PRIMITIVES REFERENCE 3.3 - 10/87
XRCN
Reset Console Inputs

Value: $A0B2
Module: MPDOSE
Syntax: 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: XCBC - Check For Break Character
Possible Errors: None
Example: DONE XRCN ;CLOSE FILES
XRCP
Read Port Cursor Position

Value: $A092
Module: MPDOSK2
Syntax: XRCP
Registers:
In     D0.W = Port #
Out    D1.L = Row
       D2.L = Column

Description:
If D0.W=0, then the current port (PRT$(A6)) is used.
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:
XCLS - Clear Screen
XPSC - Position Cursor

Possible Errors: None

Example:

```
MOVEQ.L #1,D0 ;LOOK AT PORT 1
XRCP ;READ POSITION
SWAP D1
MOVE.W D2,D1 ;D1.L=Y/X POSITION
```
XRDE
Read Next Directory Entry

Value: $A0A6
Module: MPDOSF
Syntax: 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

Example:

```
START  MOVEQ.L #0,D1  ;BEGIN WITH 1ST ENTRY
   BRA.S LOOP02
   LOOP  MOVEQ.L #-1,D1  ;READ NEXT ENTRY
      LOOP02  MOVE.W D5,R0  ;GET DISK #
            XRD   ;READ DIRECTORY ENTRY
            BNE.S ERROR  ;ERROR
            MOVE.B 12(A2),R4 ;GET FILE TYPE
```
XRDM
Dump Registers

Value: $A02A
Module: MPDOSK1
Syntax: 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. It also outputs the VBR register on 68010/20 systems.

The registers and status are not affected by this primitive.

See Also: XBUG - Debug Call
XDMP - Dump Memory From Stack
PDOS Monitor, Editor, Utilities manual

Possible Errors: None

Example:

```
MOVEM.L RL, (A7) + ;RESTORE REGISTERS
XRDM ;DUMP RESULTS
```
**XRDN**

*Read Directory Entry by Name*

**Value:** $A0A8

**Module:** MPDOSF

**Syntax:**

```
XRDN
<status error return>
```

**Registers:**

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0.B</td>
<td>D1.W</td>
</tr>
<tr>
<td>Disk number</td>
<td>Sector number in memory</td>
</tr>
<tr>
<td>MWB$</td>
<td>(A2)</td>
</tr>
<tr>
<td>File name</td>
<td>Directory entry</td>
</tr>
</tbody>
</table>

**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 (MWBS) 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:**

XFFN - Fix File Name

**Possible Errors:**

- 53 = File Not Defined
- 68 = Not PDOS Disk
- Disk errors

**Example:**

```
OPENF LEA.L FN(PC),A1 ;GET FILE NAME POINTER
XFFN ;FIX NAME IN MWB
BNE.S ERROR ;ERROR
XRDN ;READ DIRECTORY ENTRY
BNE.S ERROR ;ERROR
....
```
XRDT

Read Date

Value: $A05C
Module: MPDOSK3
Syntax: 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: XFTD - Fix Time And Date
XPAD - Pack ASCII Date
XRTM - Read Time
XUAD - Unpack ASCII Date
XUDT - Unpack Date
XUTM - Unpack Time

Possible Errors: None

Example:

```
GETD XPMC MESL ;OUTPUT PROMPT
XRDT ;GET DATE
XPLC ;OUTPUT TO SCREEN
....

MESL DC.B 'DATE=',0
```
XRFA

Read File Attributes

Value: $A0E0
Module: MPDOSF
Syntax: XRFA
<status error return>

Registers:
In  (A1) = File name
    (A2) = Directory entry
    D0.L = Disk number
    D1.L = File size (in bytes)
    D2.L = Level/attributes

Out

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:
XCFA - Close File With Attribute
XWFA - Write File Attributes
XWFP - Write File Parameters

Possible Errors:
50 = Bad File Name
53 = File Not Defined
60 = File Space Full
Disk errors

Example:

LEA.L FN(FC),A1 ;GET FILE NAME
XRFA ;READ FILE ATTRIBUTES
BNE.S ERROR ;PROBLEM
LRL.W #2,D2 ;BINARY FILE?
BCC.S PNO ;N
..... ;Y

FN DC.B 'PRGM:BIN',0
EVEN
XRFP
Read File Position

Value: $A0FE
Module: MPDOSF
Syntax:
```
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:
- XPSF - Position File
- XRWF - Rewind File

Possible Errors:
- 52 = File Not Open
- 59 = Bad File Slot
- Disk errors

Example:
```
MOVE.W D5,D1 ;GET FILE ID
XRFP ;READ FILE POSITION
BNE.S ERROR
....
```
XRLF  

Read Line From File

Value: $A0E2
Module: MPDOSF
Syntax: 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 \(\downarrow\), from the file specified by the file ID in register D1. If a \(\downarrow\) 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 \(\downarrow\) are dropped from the data stream and the \(\downarrow\) 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:
XRBFR - Read Bytes From File
XWBF - Write Bytes To File
XWLFR - Write Line To File

Possible Errors:
52 = File Not Open
56 = End of File
59 = Bad File Slot
Disk errors

Example:

```
MOVE.W D5,D1 ;GET FILE ID
LEA.L BF(PC),A2 ;GET BUFFER POINTER
XRLF ;READ LINE
BNE.S ERROR
```

```
BF  DS.B 132 ;MAXIMUM BUFFER NEEDED
```
XRNF

Rename File

Value: $A0E4

Module: MPDOSF

Syntax:

XRNF
<status error return>

Registers:

In
(A1) = Old file name
(A2) = New file name or level number

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 or level 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:

XDFL - Define File
XDLF - Delete File

Possible Errors:

50 = Bad File Name
51 = File Already Defined

Disk errors

Example:

```
LEA.L F1(PC),A1 ;GET OLD FILE NAME
LEA.L F2(PC),A2 ;GET NEW FILE NAME
XRNF ;RENAME FILE
BNE.S ERROR ;PROBLEM
MOVE.A.L A2,A1 ;POINT TO NEW NAME
LEA.L LV(PC),A2 ;GET NEW LEVEL
XRNF ;CHANGE DIRECTORY LEVEL
BNE.S ERROR

L V DC.B '10',0
F1 DC.B 'OBJECT:OLD',0
F2 DC.B 'OBJECT:NEW',0
EVEN
```

XROO
Open Random Read Only File

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

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

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.

\[
\begin{align*}
D1.W &= (\text{Disk #}) \times 256 + (\text{File slot index}) \\
D0.W &= (\text{ABOS BETD xxxx xCWD})
\end{align*}
\]

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 = Bad File Name
53 = File Not Defined
61 = File Already Open
68 = Not PDOS Disk
69 = Out of File Slots

Errors:
Disk errors

Example:

```
LEA.L HLPFN(PC),A1 ;POINT TO FILE NAME
XROO ;OPEN FILE
    BNE.S ERROR
    HELP02 MOV EA.L A6,A2 ;GET BUFFER
                    XRLF ;READ LINE
    BNE.S SHWF22
          HLPFN DC.B 'HLPTX',0
```

3.3 - 10/87 PDOS ASSEMBLY PRIMITIVES REFERENCE
**XROP**

Open Random File

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A0E8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSF</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XROP</td>
</tr>
<tr>
<td></td>
<td>&lt;status error return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>In (A1) = File name</td>
</tr>
<tr>
<td></td>
<td>Out D0.W = File attribute</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>50 = Bad File Name</td>
</tr>
<tr>
<td></td>
<td>53 = File Not Defined</td>
</tr>
<tr>
<td></td>
<td>61 = File Already Open</td>
</tr>
<tr>
<td></td>
<td>68 = Not PDOS Disk</td>
</tr>
<tr>
<td></td>
<td>69 = Out of File Slots</td>
</tr>
<tr>
<td>Example:</td>
<td>LEA.L FN(PC),A1 ;GET FILE NAME</td>
</tr>
<tr>
<td></td>
<td>XROP ;OPEN RANDOM FILE</td>
</tr>
<tr>
<td></td>
<td>BNE.S ERROR ;ERROR</td>
</tr>
<tr>
<td></td>
<td>MOVE.W D0,D5 ;SAVE TYPE</td>
</tr>
<tr>
<td></td>
<td>SWAP D5</td>
</tr>
<tr>
<td></td>
<td>MOVE.W D1,D5 ;SAVE FILE ID</td>
</tr>
<tr>
<td></td>
<td>....</td>
</tr>
<tr>
<td></td>
<td>FN DC.B 'FILENAME:EXT',0</td>
</tr>
<tr>
<td></td>
<td>EVEN</td>
</tr>
</tbody>
</table>

Uses multiple directory file search.

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.

\[
D0.W = (ABOS BETU xxxx xCWD) \\
D1.W = (Disk #) \times 256 + (File slot index)
\]

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.

Disk errors
XRPS
Read Port Status

Value: $A094
Module: MPDOSK2
Syntax: XRPS
          <status error return>
Registers:
          In    D0.W = Port number

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 ACIS.

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 Ctrl S Ctrl Q protocol is in effect, and other flags.

<table>
<thead>
<tr>
<th>portflag. = fwpil 8dcs</th>
<th>0 = Ctrl S Ctrl Q enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>1 = Ignore control character</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>2 = DTR enable</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>3 = 8-bit character enable</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>4 = Receiver interrupt disable</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>5 = Even parity enable</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>6 = (Reserved)</td>
</tr>
<tr>
<td>_ _ _ _ _ _ _ _</td>
<td>7 = (Reserved)</td>
</tr>
</tbody>
</table>

See Also: XBCP - Baud Console Port
          XSPF - Set Port Flag

Possible Errors:
66 = Bad Port/Baud Rate

Example:

```
MOVEQ.L #0, D0 ;LOOK AT CURRENT PORT
XRPS
BNE.S ERROR
BTST.B #0, D1 ;^S^Q?
BNE.S CSCQ ;Y
```
**XRSE**

Read Sector

**Value:**

$A0C2

**Module:**

MPDOSF

**Syntax:**

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:**

BIOS in *PDOS Developer's Reference Manual*

XISE - Initialize Sector

XRSZ - Read Sector Zero

XWSE - Write Sector

**Possible Errors:**

Disk errors

**Example:**

```
CLR.W D0    ;SELECT DISK #0
MOVEQ.L #2,D1    ;SELECT SECTOR 2
LEA.L BUFF(PC),A2 ;POINT TO BUFFER
XRSE        ;READ INTO BUFFER
BNE.S XERR    ;ERROR
....
XERR XERR    ;DISK ERROR
BUFFER DS.B 256 ;BUFFER
```
**XRSR**

**Read Status Register**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A042</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XRSR</td>
</tr>
<tr>
<td>Registers:</td>
<td>Out D0.W = 68000 status register</td>
</tr>
<tr>
<td>Description:</td>
<td>The READ STATUS REGISTER primitive allows you to read the 68000 status register. Of course, this is equivalent to the &quot;MOVE.W SR,Dx&quot; 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.</td>
</tr>
</tbody>
</table>
| See Also: | XLSR - Load Status Register  
|           | XSUP - Enter Supervisor Mode  
|           | XUSP - Return to User Mode |
| Possible Errors: | None |
| Example: | XRSR ;READ SR  
|           | ANDI.W #$0700,DO |

**Possible Errors:**

None
**XRST**

**Reset Disk**

**Value:**
$A0B4

**Module:**
MPDOSF

**Syntax:**
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:**
XCFA - Close File With Attribute
XCLF - Close File

**Possible Errors:**
None

**Example:**

```
DONE     MOVEQ.L #-1,D1  ;CLOSE ALL TASK FILES
XRST
    ...  

    MOVE.W D5,D1  ;PREPARE TO REMOVE DISK
XRST  ;CLOSE ALL FILES
    ...  
    ;REMOVE DISK
```
XRSZ

Read Sector Zero

Value: $A0C4
Module: MPDOSF
Syntax: 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: BIOS in PDOS Developer's Reference Manual
          XISE - Initialize Sector
          XRSE - Read Sector
          XWSE - Write Sector

Possible Errors: Disk errors

Example:

    MOVEQ.L #1,D0 ;SELECT DRIVE 1
    XRSZ ;READ HEADER
    BNE.S ERROR
    XPBC ;PRINT DISK NAME
**XRTE**

**Return From Interrupt**

**Value:**
$A044$

**Module:**
MPDOSK1

**Syntax:**
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. It also allows a return from an interrupt to awaken a specific task regardless of higher priority tasks. To signal XRTE to return to a specific task, the interrupt routine sets the task number into byte TQUX.(AS) in the system SYRAM.

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

**Example:**

```
.;PROCESS INTERRUPT
MOVEQ.L #66,D1
XSEV ;SET EVENT 66
XRTE ;RETURN FROM INTERRUPT
```
## XRTM

**Read Time**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A05E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK3</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XRTM</td>
</tr>
</tbody>
</table>
| 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:    | XFTD - Fix Time And Date
|              | XPAD - Pack ASCII Date
|              | XRTD - Read Date
|              | XUAD - Unpack ASCII Date
|              | XUDT - Unpack Date
|              | XUTM - Unpack Time |
| Possible Errors: | None |
| Example:     | GETD XPMC MES1 ;OUTPUT PROMPT
|              | XRTM ;GET TIME
|              | XPLC ;OUTPUT TO SCREEN
|              | ... |
|              | MES1 DC.B 'TIME=','0 |
| **Value:** | $A034 |
| **Module:** | MPDOSK1 |
| **Syntax:** | 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:** | XFTD - Fix Time And Date  
  XPAD - Pack ASCII Date  
  XRDT - Read Date  
  XRTM - Read Time  
  XUAD - Unpack ASCII Date  
  XUDT - Unpack Date  
  XUTM - Unpack Time |
| **Possible Errors:** | None |
XRTS

Read Task Status

Value:          $A012
Module:        MPDOSK1
Syntax:        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

Description:

If D0.W=-1, then the current task number is returned in D1.L.

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:
XSTP - Set/Read Task Priority

Possible Errors:
None

Example:

```
WAIT    MOVEQ.L #2,D0 ;WAIT TO TASK 0
XRST    ; TO DIE
BNE.S  WAIT  ;STILL GOING
...      ;DONE
```
XRWF
Rewind File

Value: $A0EA
Module: MPDOSF
Syntax: 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: XPSF - Position File
XRFP - Read File Position
Possible Errors: 52 = File Not Open
59 = Bad File Slot
Disk errors
Example:

```
REWIND MOVE.W D5,D1 ;GET FILE ID
XRWF ;REWIND FILE
BNE.S ERROR ;PROBLEM
```
**XSEF**

**Set Event Flag With Swap**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XSEF</td>
</tr>
<tr>
<td></td>
<td>&lt;status return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>In D1.B = Event (+=Set(1), -=Clear(0))</td>
</tr>
<tr>
<td></td>
<td>Out SR = NE...,Set</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An XSWP is automatically executed after the event is set or cleared. 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 cleared.

**Description:**

The SET EVENT FLAG WITH SWAP primitive sets or clears an event flag bit. The event number is specified in data register D1.B and is modulo 128. If the content of register D1.B is positive (1 to 127, $01 to $7F), then the event bit is set (1). If D1.B is negative (-1 to -127, $FF to $81), the bit is cleared (=0). Event 128 can only be cleared. (It is set by the delay event list.)

If the event is 128 ($80) then the task’s local event is cleared. Event zero ($00) is illegal to use. The status of the event bit prior to changing the event is returned in the status register. If the event was cleared, then the “EQ” status is returned; otherwise, if the event was set, then a “NE” status is returned. Also, an immediate context switch occurs thus scheduling any higher priority task pending on that event.

<table>
<thead>
<tr>
<th>Four types of event flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-63 = Software</td>
</tr>
<tr>
<td>64-80 = Software self clearing</td>
</tr>
<tr>
<td>81-127 = System</td>
</tr>
<tr>
<td>128 = Local to task</td>
</tr>
</tbody>
</table>
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 = 1/5 second event
113 = 1 second event
114 = 10 second event
115 = 20 second event
116 = TTA active
117 =
118 = Printer
119 = Disk
120 = Level 2 lock
121 = Level 3 lock
122 = Batch event
123 = Spooler event
124 =
125 =
126 = Error message disable
127 = System utility
128 = Local

See Also:
XDEV - Delay Set/Clear Event
XSEV - Set Event Flag
XSUI - Suspend Until Interrupt
XTEF - Test Event Flag

Possible Errors:
None

Example:

```
MOVEQ.L #30,D1 ; SET EVENT 30
XSEF ; SET EVENT
....

MOVEQ.L #-35,D1 ; CLEAR EVENT 35
XSEF ; SET EVENT
....
```
XSEV
Set Event Flag

Value: $A046
Module: MPDOSKI
Syntax: XSEV
<status return>

Registers:
In  D1.B = Event (+=Set(1), -=Clear(0))
Out  SR = NE....Set
     EQ....Reset

Event 128 is local to each task.
If D1.B is positive, then the event is set.
If D1.B is negative, then the event is reset.

The SET EVENT FLAG primitive sets or clears an event flag bit. The event number is specified in data register D1.B and is modulo 128. If the content of register D1.B is positive (1 to 127, $01 to $7F), then the event bit is set (=1). If D1.B is negative (-1 to -127, $FF to $811), the bit is cleared (=0). Event 128 can only be cleared. (It is set by the delay event list.) Event zero ($00) is illegal to use. If the event is 128 ($80) then the task’s local event is cleared. The status of the event bit prior to changing the event is returned in the status register. If the event was cleared, then the “EQ” status is returned; otherwise, if the event was set, then a “NE” status is returned. A context switch DOES NOT occur with this call making it useful for interrupt routines outside the PDOS system.

Four types of event flags:
1-63 = Software
64-80 = Software self clearing
81-127 = System
128 = Local to task

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
XSEV - Set Event Flag

See Also:
- XDEV - Delay Set/Reset Event
- XSEV - Set Event Flag
- XSUI - Suspend Until Interrupt
- XTEF - Test Event Flag

Possible Errors:
None

Example:

```
MOVEQ.L #30,D1 ;SET EVENT 30
XSEV ;SET EVENT
....

MOVEQ.L #-35,D1 ;CLEAR EVENT 35
XSEV ;SET EVENT
....
```
XSMP

Send Message Pointer

Value: $A002
Module: MPDOSK1
Syntax: 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
     D0.L = Error number 62 if message pointer error
Description:
The SEND MESSAGE POINTER primitive sends a 32-bit message to the mes-
sgage slot specified by data register D0.B. Address register A1 contains the mes-

If there is still a message pending, then the primitive immediately returns with
status set“Not Equal” and D0.L returns the error number 62. Otherwise, the mes-

The primitive XSMP is only valid for message slots 0 through 15. (This is be-
cause of current event limitations.)

See Also:
XGMP - Get Message Pointer
XGTM - Get Task Message
XKTM - Kill Task Message
XSTM - Send Task Message
Possible
Errors:
62 = Bad Message Ptr Call
Example:

```
LEA.L MESS(PC),A1 ;LOAD ADDRESS OF MESS INTO A1
AGAIN MOVE.B $5,DO ;POINT TO MESSAGE SLOT #5
XGMP ;SEND MESSAGE TO SLOT 5
BEQ.S AROUND ;MESSAGE SENT
XWSP
BRA.S AGAIN ;MESSAGE PENDING, SO WAIT AWHILE
AROUND . .
MESS DC.B $0A,$0D,‘HELLO PDOS USERS’,0
```
XSOE
Suspend on Physical Event

Value: $A112
Module: MPDOSK1
Syntax: 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.L = -1 if awaken on event 0; 1 if awaken on event 1

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 on the bit. The following bits are defined:

<table>
<thead>
<tr>
<th>Bit number</th>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>x x x x x x x x x x x x x x x x</td>
</tr>
<tr>
<td>S</td>
<td>x x x x B B</td>
</tr>
<tr>
<td>BBB</td>
<td>The 680x0 bit number to use as an event</td>
</tr>
</tbody>
</table>

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

Example:

```assembly
MOVE.L #$80800081,D1 ;SET DESCRIPTORS
LEA.L PEV(PC),A0 ;GET PEV ADDRESS
MOVEA.L A0,A1 ;COPY FOR EV1
MOVE.L #100,D0 ;SET TIMEOUT
BCLR.B D0,(A1) ;CLEAR TIMEOUT EVO
XDPE ;START TIMER
XSOE ;SUSPEND

PEV DC.W 0
```

PDOS ASSEMBLY PRIMITIVES REFERENCE 3.3 - 10/87
XSOP

Open Sequential File

Value: $A0EC
Module: MPDOSF
Syntax: XSOP
<status error return>
Registers:
   In  (A1) = File name
   Out D0.W = File attribute
         D1.W = File ID

Description:
Uses multiple directory file search.

The OPEN SEQUENTIAL FILE primitive opens a file for sequential access by assigning the file to an area of system memory called a file slot and returning a file ID and file type to the calling program. Thereafter, the file is referenced by the file ID and not by the file name.

The file ID (returned in register D1) is a 2-byte number. The left byte is the disk number and the right byte is the file slot index. The file attribute is returned in D0.

D0.W = (ABOS BETD xxxx xCDW)
D1.W = (Disk #) x 256 + (File slot index)

The END-OF-FILE marker on a sequential file is changed whenever data is written to the file. All data transfers are buffered through a channel buffer; data movement to and from the disk is by full sectors.

The file slots are allocated beginning with slot 32 down to slot 1.

Possible Errors:
50 = Bad File Name
53 = File Not Defined
61 = File Already Open
68 = Not PDOS Disk
69 = Out of File Slots
Disk errors
**Example:**

```
LEA.L FN(PC),A1 ;GET FILE NAME
XSOP ;OPEN SEQUENTIAL FILE
BNE.S ERROR ;ERROR
MOVE.W D0,D5 ;SAVE TYPE
SWAP D5
MOVE.W D1,D5 ;SAVE FILE ID
....
FN DC.B 'FILENAME:EXT',0
EVEN
```
XSPF

Set Port Flag

Value: $A09A

Module: MPDOSK2

Syntax: XSPF
         <status error return>

Registers:

In       D0.W = Port number
         D1.B = Port flag (fwpi8dcs)

Out      D1.B = Old port flag

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: XBCP - Baud Console Port
          XRPS - Read Port Status

Possible
Errors: 66 = Bad Port/Baud Rate

Example:

MOVEQ.L #0,D0 ;SELECT CURRENT
MOVEQ.L #1,D1 ;"S"Q
XSPF
### XSTM

**Send Task Message**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XSTM &lt;status error return&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td>D0.B = TASK NUMBER (A1) = MESSAGE</td>
</tr>
<tr>
<td>Description:</td>
<td>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. D0 = -1 sends message to parent 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.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XGMP - Get Message Pointer XGTM - Get Task Message XKTM - Kill Task Message XSMP - Send Message Pointer XSTM - Send Task Message</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>78 = Msg Buffer Full</td>
</tr>
<tr>
<td>Example:</td>
<td>TERR LEA.L ERM(PC),A1 ;RETURN MESSAGE ST.B D0 ;SEND TO PARENT XSTM ;SEND, ERROR? BNE.S ERROR ;Y XEXT ;N, QUIT</td>
</tr>
</tbody>
</table>
XSTP

Set/Read Task Priority

Value: $A03C
Module: MPDOSK1
Syntax: XSTP
<status error return>
Registers:

<table>
<thead>
<tr>
<th>In</th>
<th>D0.B = Task #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D1.W = Task time/Task priority</td>
</tr>
<tr>
<td>Out</td>
<td>D1.B = Task priority (If D1.B was 0)</td>
</tr>
</tbody>
</table>

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.

Example:

```
MOVEQ.L #1,DO ;CURRENT TASK
MOVEQ.L #0,D1 ;SET TO READ
XSTP ;READ TASK PRIORITY
BNE.S ERROR
MOVE.B D1,SV(A2)
;

MOVEQ.L #1,DO ;SELECT CURRENT
MOVEQ.L #100,D1 ;SET TO WRITE
XSTP ;SET TASK PRIORITY
BNE.S ERROR
```
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 clears (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 cleared), 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 determine 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 cleared 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:
- XDEV - Delay Set/Clear Event
- XDPE - Delay on Physical Event
- XSEF - Set Event Flag With Swap
- XSEV - Set Event Flag
- XSOE - Suspend on Physical Event
- XTEF - Test Event Flag

Possible Errors:
None
XSUI - Suspend Until Interrupt

Example:

```
GETC XGCC ;CHARACTER?
BNE.S GETC2 ;Y
MOVEQ.L #100,D0 ;N, GET DELAY
MOVEQ.L #128,D1 ;USER LOCAL EVENT
XDEV ;DELAY 128 1 SECOND
BNE.S GETC ;FULL
LSL.W #8,D1 ;GET 128/(PORT+96)
MOVE.B #96,D1
ADD.B PRT$(A6),D1
XSUI ;SUSPEND
CMP.B D0,D1 ;CHARACTER EVENT?
BEQ.S GETC ;Y
```
XSUP
Enter Supervisor Mode

Value:  $A02C
Module:  MPDOSK1
Syntax:  XSUP
Registers:  None
Description:
The ENTER SUPERVISOR MODE primitive moves your current task from user mode to supervisor mode. Take care 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.

Exit to user mode by executing a "ANDI.W #$DFFF,SR" instruction or the XUSP primitive.

See Also:
XLSR - Load Status Register
XRSR - Read Status Register
XUSP - Return To User Mode

Possible Errors:
None

Example:

```
* EQU $FFFFCE01 ;I/O PORT
OUT XSUP ;ENTER SUPERVISOR
MOVE.B D0,P1 ;OUTPUT
ANDI.W #$DFFF,SR ;MOVE TO USER
RTS ;RETURN
```
## XSWP

### Swap to Next Task

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XSWP</td>
</tr>
<tr>
<td>Registers:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>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.)</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>None</td>
</tr>
<tr>
<td>Example:</td>
<td>LOOP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| | | LOOP02 | .....


**Value:** $A0B6

**Module:** MPDOSF

**Syntax:**
- 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
Example:

```
CLR.L D0 ;SELECT DISK #0
XSZF ;GET DISK SIZE
   BNE.S ERROR ;ERROR
CLR.L D1
MOVE.W D7,D1
XCBM SPM1 ;OUTPUT FREE
XPLC ;PRINT
SWAP D7
MOVE.W D7,D1
XCBM SPM2 ;OUTPUT LARGEST
XPLC ;CONTIGUOUS BLOCK
XTAB 20 ;TAB TO COLUMN 20
MOVE.W D6,D1
XCBM SPM3 ;OUTPUT USED
XPLC ;PRINT
SWAP D6
MOVE.W D6,D1
XCBM SPM4 ;OUTPUT ALLOCATED
XPLC ;PRINT
XEXT
```

* SPM1 DC.B $0A,$0D,'FREE:',0
* SPM2 DC.B ',',0
* SPM3 DC.B 'USED:',0
* SPM4 DC.B 'ALLOCATED',0
EVEN
<table>
<thead>
<tr>
<th><strong>Value:</strong></th>
<th>$A090</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module:</strong></td>
<td>MPDOSK2</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td>XTAB &lt;column&gt;</td>
</tr>
<tr>
<td><strong>Registers:</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Possible Errors:</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### Example:

```
XPMC MES1 ;OUTPUT HEADER
XTAB 30 ;MOVE TO COLUMN 30
.....
```
**XTEF**

*Test Event Flag*

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A01A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
</tbody>
</table>
| Syntax: | XTEF  
  <status return> |
| Registers: | In  
  D1.B = Event number (+=1-127, -=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 modulo 128. Event 128 is local to each task. |
| See Also: | XDEV - Delay Set/Clear Event  
XSEF - Set Event Flag With Swap  
XSEV - Set Event Flag  
XSUI - Suspend Until Interrupt |
| Possible Errors: | None |
| Example: | MOVEQ.L #30,D1 ;EVENT 30  
XTEF ;TEST EVENT FLAG  
BNE.S EVENT ;EVENT = .TRUE.  
..... ;EVENT = .FALSE. |
XTLP
Translate Logical to Physical Event

Value: $A110
Module: MPDOSK1
Syntax: 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 PDOS 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 PDOS logical event is one of the 128 events maintained by the PDOS 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 680x0 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. 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.
XTLP - Translate Logical to Physical Event

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

Example:

```
MOVE.L #128,D1   ;GET LOGICAL EVENT
MOVE.L #100,D0   ;SET TIMEOUT
XDEV            ;START TIMER
LSL.W           #8,D1 ;MAKE EVENT 1
XTLP            ;TRANSLATE TO PHYSICAL
MOVE.W #$8080,D1 ;BIT 0 SET AND TOGGLE
LEA.L PEV(PC),A0 ;GET PEV ADDRESS
XSOE            ;SUSPEND UNTIL BIT 0 OF PEV
        ;PEV IS A 1
...            
PEV            DC.W 0
```
XUAD
Unpack ASCII Date

Value: $A036
Module: MPDOSK3
Syntax: XUAD
Registers:
  In  D1.W = (Year*16+Month)*32+Day
       (YYYY YYYY 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 slashes.

Note: XUAD does not check for a valid date and hence, strange strings could result. In-
valid months are replaced by“???.”

See Also: XFTD - Fix Time and Date
          XPAD - Pack ASCII Date
          XRDT - Read Date
          XRTM - Read Time
          XUDT - Unpack Date
          XUTM - Unpack Time

Possible Errors: None
**XUDT**

**Unpack Date**

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A060</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK3</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XUDT</td>
</tr>
</tbody>
</table>
| 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:     | XFTD - Fix Time and Date  
               | XPAD - Pack ASCII Date  
               | XRDT - Read Date  
               | XRTM - Read Time  
               | XUAD - Unpack ASCII Date  
               | XUTM - Unpack Time |
| Possible Errors: | None |
| Example:      | XFTD       ;FIX TIME & DATE  
               | XUDT       ;UNPACK DATE  
               | XPLC       ;PRINT 'MN/DY/YR'  
               | ...         |
XULF
Unlock File

Value: $A0EE
Module: MPDOSF
Syntax: 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: XULKF - Lock File
Possible Errors:
S2 = File Not Open
59 = Bad File Slot
Disk errors
Example:

<table>
<thead>
<tr>
<th>MOVE.W D5,D1 ;GET FILE ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>XULF</td>
</tr>
<tr>
<td>;UNLOCK FILE</td>
</tr>
<tr>
<td>BNE.S ERROR</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
XULT

Unlock Task

Value: $A016
Module: MPDOSK1
Syntax: 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: XLKT - Lock Task

Possible Errors: None

Example:

```
* XLKT ;LOCK TASK WHILE WAITING
  LOOP TST.B LMEM ;CONDITION MET?
  BNE.S LOOP ;N, WAIT
  CLR.B OMEM ;Y, RESET
  XULT ;UNLOCK TASK NOW
```
XUSP
Return to User Mode

Value: $A008
Module: MPDOSK1
Syntax: 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: XLSR - Load Status Register
           XSUP - Enter Supervisor Mode

Possible Errors: None

Example:

<table>
<thead>
<tr>
<th>P1</th>
<th>EQU $FFFFCEO1 ; I/O PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>XUSP</td>
</tr>
<tr>
<td></td>
<td>; ENTER SUPERVISOR</td>
</tr>
<tr>
<td></td>
<td>MOVE.B D0,P1</td>
</tr>
<tr>
<td></td>
<td>; OUTPUT</td>
</tr>
<tr>
<td></td>
<td>XUSP</td>
</tr>
<tr>
<td></td>
<td>; RETURN TO USER</td>
</tr>
<tr>
<td>RTS</td>
<td>; RETURN</td>
</tr>
</tbody>
</table>
XUTM

Unpack Time

Value: $A062

Module: MPDOSK3

Syntax: 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:

XFTD - Fix Time and Date
XPAD - Pack ASCII Date
XRDT - Read Date
XRTM - Read Time
XUAD - Unpack ASCII Date
XUDT - Unpack Date

Possible Errors:

None

Example:

```
XFTD   ;GET SYSTEM TIME
MOVE D0,D1
XUTM   ;CONVERT TO STRING
XPLC   ;PRINT TIME
```
XVEC

Set/Read Exception Vector

<table>
<thead>
<tr>
<th>Value:</th>
<th>$A116</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module:</td>
<td>MPDOSK1</td>
</tr>
<tr>
<td>Syntax:</td>
<td>XVEC</td>
</tr>
<tr>
<td>Registers:</td>
<td></td>
</tr>
<tr>
<td>In</td>
<td>D0.W = Exception number (#2-255)</td>
</tr>
<tr>
<td></td>
<td>(A0) = New exception service routine (0=read only)</td>
</tr>
<tr>
<td>Out</td>
<td>(A0) = Old service routine</td>
</tr>
<tr>
<td>Description:</td>
<td>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.</td>
</tr>
<tr>
<td>See Also:</td>
<td>XDTV - Define Trap Vectors</td>
</tr>
<tr>
<td>Possible Errors:</td>
<td>None</td>
</tr>
</tbody>
</table>

**Example:**

```assembly
START MOVEQ.L #5,DO ;ZERO
LEA.L ZDIV(PC),AO ;GET NEW SYSTEM ZERO DIV VEC
XVEC ;SET
DIVU.W #0,DO ;ZERO DIV ERROR
XEXT ;WILL EXECUTE AFTER ZDIV EXCEPT

* ZDIV EXCEPTION HANDLER
* ZDIV XPMC M1 ;ZERO DIV EXCEPTION
MOVEQ.L #5,DO
XVEC ;SET
RTE ;RETURN FROM EXCEPTION

M1 DC.B $0A,$0D,'ZERO DIVIDE EXCEPTION',0
* END START
```

Refer to the *Installation and Systems Management* guide for a list of user vectors that are implemented on your hardware. Changing vectors that are in use may cause the system to crash.
XWBF
Write Bytes to File

Value: $A0F0
Module: MPDOSF
Syntax: 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:
XRBF - Read Bytes From File
XRLF - Read Line From File
XWLF - Write Line To File

Possible Errors:
52 = File Not Open
55 = Too Few Contiguous Sectors
58 = File Delete or Write Protected
59 = Bad File Slot
60 = File Space Full
Disk errors

Example:

```
MOVE.L #252,D0 ;WRITE FULL SECTOR
MOVE.W D5,D1 ;GET ID
LEA.L BF(PC),A2 ;GET BUFFER ADDRESS
XWBF ;WRITE TO FILE
BNE.S ERROR

* * *

BF DS.B 256 ;SECTOR BUFFER
```
**XWDT**

**Write Date**

**Value:** $A064

**Module:** MPDOSK3

**Syntax:** 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

**Example:**

```assembly
MOVEQ.L #12,D0 ;SET DATE TO 12/25/80
MOVEQ.L #25,D1
MOVEQ.L #83,D2
XWDT ;SET DATE
```
XWFA

Write File Attributes

Value: $A0F2

Module: MPDOSF

Syntax: XWFA
<status error return>

Registers:
In (A1) = File name
(A2) = ASCII file attributes

(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
BK - 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:
XCFA - Close File With Attribute
XRFA - Read File Attributes
XWFP - Write File Parameters

Possible Errors:
50 = Bad File Name
53 = File Not Defined
54 = Bad File Attribute
Disk errors

Example:

LEA.L FN(PC),A1 ;GET FILE NAME
LEA.L PF(PC),A2 ;SET BINARY & PROTECTED
XWFA ;SET
BNE.S ERROR
****

FN DC.B 'DATA:BIN',0
PF DC.B 'BN**',0
EVEN
**XWFP**

**Write File Parameters**

**Value:**
$A0FC$

**Module:**
MPDOSF

**Syntax:**
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 = ORed 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:**
- XCFA - Close File With Attribute
- XRFA - Read File Attributes
- XWF A - Write File Attributes

**Possible Errors:**
- 50 = Bad File Name
- 53 = File Not Defined
- Disk errors

**Example:**

```assembly
LEA.L  FN(PC),A1   ;GET FILE NAME
XRFA   ;READ FILE ATTRIBUTES
BNE.S  ERROR    ;ERROR
ADDA.W #20,A2   ;POINT TO
MOVEM.L (A2),D5-D7 ;SAVE PARAMETERS
...

MOVE.L D5,D0
MOVE.L D6,D1
MOVE.L D7,D2
LEA.L  FN(PC),A1   ;GET FILE NAME
XWFP   ;UPDATE FILE PARAMETERS
BNE.S  ERROR
....

FN  DC.B  'DATA:BIN',0
EVEN
```
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.

Possible Errors:

52 = File Not Open
55 = Too Few Contiguous Sectors
58 = File Writ/Del Prot
59 = Bad File Slot
60 = File Space Full

Disk errors

Example:

```
MOVE.W D5,D1 ;GET FILE ID
LEA.L LB(PC),A2 ;GET LINE
XWLF ;WRITE LINE
BNE.S ERROR ;ERROR
....
LB DC.B $0A,$0D,'NO DIAGNOSTICS',0
EVEN
```
XWSE
Write Sector

Value: $A0C6
Module: MPDOSF
Syntax: 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: BIOS in PDOS Developer's Reference Manual
XISE - Initialize Sector
XRSE - Read Sector
XRSZ - Read Sector Zero

Possible Errors: Disk errors

Example:

```
CLR.L D0 ;WRITE TO DISK #0
MOVEQ.L #10,D2 ;WRITE TO SECTOR #10
LEA.L BUF(PC),A2 ;GET BUFFER ADDRESS
XWSE ;WRITE
BNE.S ERROR ;PROBLEM
....
BUF DS.B 256 ;DATA BUFFER
```
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.

Example:

```
MOVEQ.L $23, D0 ;SET TIME TO 23:59:59
MOVEQ.L $59, D1
MOVEQ.L $59, D2
XWTM ;SET SYSTEM TIME
```
Zero File

Value: $A0F6
Module: MPDOSF
Syntax: 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: XDFL - Define File
XDLF - Delete File
Possible Errors:
50 = Bad File Name
61 = File Already Open
68 = Not PDOS Disk
Disk errors
Example:

```
LEA.L FN(PC),A1 ;POINT TO FILE
XZFL ;ZERO FILE
BNE.S ERROR
*****
FN DC.B 'FILE:SR',0
EVEN
```
Index

Character
get C., 58
get C. conditional, 55 - 56
get port C., 57
place C. in port buffer, 89
put C. raw, 90
put C. to console, 87
Check
for break character, 19
for break or pause, 23
for file altered, 51
Clear
delay C. event, 37
file, 159
screen, 32
Close
file, 31
file with attribute, 26
Column
  tab to C., 141
Command
  push C. to buffer, 86
Conditional
  get character, 55
  get character C., 56
Console
  baud C. port, 13
  I/O calls, 6
  put buffer to C., 85
  put character to C., 87
  put CRLF to console, 88
  put data to C., 91
  put encoded line to C., 92
  put encoded message to C., 94
  put line to C., 95
  put message to C., 96
  put space to C., 100
  reset C. inputs, 102
Constants
  system C., 2
Convert
  ASCII to binary, 25
  binary to decimal, 20
  binary to hex, 21
  binary to hex in buffer, 29
  to decimal in buffer, 24
  to decimal with message, 22
Copy
  file, 33
Create
  task block, 34

Character
get C., 58
get C. conditional, 55 - 56
get port C., 57
place C. in port buffer, 89
put C. raw, 90
put C. to console, 87
Check
for break character, 19
for break or pause, 23
for file altered, 51
Clear
delay C. event, 37
file, 159
screen, 32
Close
file, 31
file with attribute, 26
Column
  tab to C., 141
Command
  push C. to buffer, 86
Conditional
  get character, 55
  get character C., 56
Console
  baud C. port, 13
  I/O calls, 6
  put buffer to C., 85
  put character to C., 87
  put CRLF to console, 88
  put data to C., 91
  put encoded line to C., 92
  put encoded message to C., 94
  put line to C., 95
  put message to C., 96
  put space to C., 100
  reset C. inputs, 102
Constants
  system C., 2
Convert
  ASCII to binary, 25
  binary to decimal, 20
  binary to hex, 21
  binary to hex in buffer, 29
  to decimal in buffer, 24
  to decimal with message, 22
Copy
  file, 33
Create
  task block, 34
Index - cont'd.

Cursor
position C., 97
read port C. position, 103

Errors
PDOS E. listing, 9
Event
delay on physical E., 42
delay set/clear E., 37
suspend on physical E., 129
translate logical E. to physical E., 143
Event Flag
set E.F., 126
set E.F. with swap, 124
test E.F., 142
Exception
set/read E. vector, 151
Execute
PDOS call D7.W, 47
Exit
to monitor, 49
to monitor with command, 50
External
PDOS symbols, 2

D
Data
conversion calls, 6
put D. to console, 91
Date
fix D., 53
pack ASCII D., 84
read D., 107
unpack ASCII D., 145
unpack D., 146
write D., 153
Debug
call, 17
Decimal
convert binary to D., 20
convert to D. in buffer, 24
correct to D. with message, 22
Define
file, 38
trap vectors, 44
Delay
on physical event, 42
set/clear event, 37
Delete
file, 40
Directory
build file D. list, 15
list file D., 81
read D. entry by name, 106
read next D. entry, 104
Disk
access calls, 8
get D. size, 139
reset D., 117
Dump
memory from stack, 41
registers, 105

E
Encoded
put E. line to console, 92
put E. message to console, 94
Enter
supervisor mode, 137
Error
load E. register, 75
return E. D0 to monitor, 46
return status E., 4
trapping, 4

File
altered check, 51
append F., 12
build F. directory list, 15
chain F., 28
close, 31
close F. with attribute, 26
copy, 33
define F., 38
delete, 40
list F. directory, 81
load F., 73
lock F., 78
look for name in file slots, 76
management calls, 7
open random F., 113
open random read only F., 112
open sequential F., 130
open shared random F., 82
position F., 99
read bytes from F., 101
read F. attributes, 108
read F. position, 109
read line from F., 110
rename F., 111
rewind F., 123
support calls, 7
unlock, 147
write bytes to F., 152
write F. attributes, 154
write F. parameters, 155
write line to F., 156
zero F., 159
Filename
   fix F., 52
Fix
   filename, 52
time and date, 53
Flag
   set port F., 132
Flush
   buffers, 51
Format
   assembly F., 3
   of source files, 3
Free
   user memory, 54

Get
   character, 58
calendar conditional, 55 - 56
disk size, 139
line in buffer, 59
line in monitor buffer, 61
line in user buffer, 62
memory limits, 64
message pointer, 65
next parameter, 66
port character, 57
task message, 68
user memory, 69

Hex
   convert binary to H., 21
   convert binary to H. in buffer, 29

I/O
   console I/O calls, 6
Initialize
   sector, 70
Input
   reset console I., 102
Interrupt
   return from I., 119
   suspend until I., 135

Kill
   task, 71
task message, 72

Limits
   get memory L., 64
Line
   get L. in buffer, 59
   get L. in monitor buffer, 61
   get L. in user buffer, 62
   put encoded L. to console, 92
   put L. to console, 95
   read L. from file, 110
   write L. to file, 156
Line Feed
   put LF to console, 88
List
   file directory, 81
Load
   error register, 75
   file, 73
   status register, 80
Lock
   file, 78
task, 79
Logical
   translate L. event to physical, 143
Look
   for name in file slots, 76

Manual
   conventions of this M., 1
Memory
   dump M. from stack, 41
   free user M., 54
   get M. limits, 64
   get user M., 69
Message
   get M. pointer, 65
   get task M., 68
   kill task M., 72
   put encoded M. to console, 94
   put M. to console, 96
   send M. pointer, 128
   send task M., 133
Monitor
   exit to M., 49
   exit to M. with command, 50
   get line in M. buffer, 61

Name
   look for N. in file slots, 76
Next
get N. parameter, 66

O
Open
random file, 113
random read only file, 112
sequential file, 130
shared random file, 82

P
Pack
ASCII date, 84
Parameter
get next P., 66
Parameters
write file P., 155
Pause
check for P., 23
Physical
translate logical to P. event, 143
Pointer
get message P., 65
send message P., 128
Port
baud console P., 13
get P. character, 57
place character in P. buffer, 89
read P. cursor position, 103
read P. status, 114
set P. flag, 132
Position
cursor, 97
file, 99
read file P., 109
Priority
set/read task P., 134
Push
command to buffer, 86
Put
buffer to console, 85
character raw, 90
cracter to console, 87
CRLF to console, 88
data to console, 91
encoded line to console, 92
encoded message to console, 94
line to console, 95
message to console, 96
space to console, 100

R
Random
open R. file, 113
open R. read only file, 112
open shared R. file, 82
Raw
put character R., 90
Read
bytes from file, 101
date, 107
directory entry by name, 106
file attributes, 108
file position, 109
line from file, 110
next directory entry, 104
open random R. only file, 112
port cursor position, 103
port status, 114
sector, 115
sector zero, 118
status register, 116
task priority, 134
task status, 122
time, 120
time parameters, 121
Register
load error R., 75
load status R., 80
read status R., 116
usage, 1
Registers
dump R., 105
using assembly R., 4
Rename
file, 111
Reset
console inputs, 102
disk, 117
Return
error D0 to monitor, 46
from interrupt, 119
to user mode, 149
Rewind
file, 123

S
Save
68881 enable, 11
Screen
clear, 32
Sector
initialize S., 70
Index - cont'd

Read
- send T. message, 133
- set/read T. priority, 134
- swap to next T., 138
- unlock T., 148

Send
- message pointer, 128
- task message, 133

Sequential
- open S. file, 130

Set
- delay S. event, 37
- event flag, 126
- event flag with swap, 124
- exception vector, 151
- port flag, 132
- task priority, 134

Shared
- open S. random file, 82

Size
- get disk S., 139

Slot
- look for name in file S., 76

Source
- file format, 3

Space
- put S. to console, 100

Stack
- dump memory from S., 41

Status
- load S. register, 80
- read port S., 114
- read S. register, 116
- read task S., 122
- registers, 4

Supervisor
- enter S. mode, 137

Suspend
- on physical event, 129
- until interrupt, 135

Swap
- to next task, 138

System
- calls, 5
- support calls, 6
- variables, 2

Write
- bytes to file, 152
- date, 153
- file attributes, 154
- file parameters, 155
- line to file, 156
- sector, 157
- time, 158

Unlock
- file, 147
- task, 148

Unpack
- ASCII date, 145
- date, 146
- time, 150

User
- free U. memory, 54
- get line in U. buffer, 62
- get U. memory, 69
- return to U. mode, 149

Variables
- system V., 2

Vector
- set/read exception V., 151

Vectors
- define trap V., 44

Tab
- to column, 141

Task
- create T. block, 34
- get T. message, 68
- kill T., 71
- kill T. message, 72
- lock T., 79
| X | X881, 11   | XLKT, 79  |
|   | XAPF, 12  | XLSR, 80  |
|   | XBCP, 13  | XLST, 81  |
|   | XBFL, 15  | XNOP, 82  |
|   | XBUG, 17  | XPAD, 84  |
|   | XCBC, 19  | XPBC, 85  |
|   | XCBD, 20  | XPCB, 86  |
|   | XCBH, 21  | XPC, 89   |
|   | XCBM, 22  | XPCR, 90  |
|   | XCBP, 23  | XPDC, 91  |
|   | XCBX, 24  | XPEL, 92  |
|   | XCDB, 25  | XPES, 94  |
|   | XCFA, 26  | XPLC, 95  |
|   | XCHF, 28  | XPMC, 96  |
|   | XCHX, 29  | XPS, 97   |
|   | XCLF, 31  | XPSF, 99  |
|   | XCLS, 32  | XPSP, 100 |
|   | XCPY, 33  | XRB, 101  |
|   | XCTB, 34  | XRCN, 102 |
|   | XDEV, 37  | XRP, 103  |
|   | XDFF, 38  | XRD, 104  |
|   | XDLF, 40  | XRDG, 105 |
|   | XDMP, 41  | XRDN, 105 |
|   | XDPF, 42  | XRD, 107  |
|   | XDFT, 44  | XRF, 108  |
|   | XERR, 46  | XRF, 109  |
|   | XEXC, 47  | XRLF, 110 |
|   | XEXG, 49  | XRN, 111  |
|   | XEXZ, 50  | XRO, 112  |
|   | XFAC, 51  | XRO, 113  |
|   | XFBB, 51  | XRPA, 114 |
|   | XFFN, 52  | XRS, 115  |
|   | XFTD, 53  | XRSR, 116 |
|   | XFTD, 53  | XRS, 117  |
|   | XFRM, 54  | XRSR, 117 |
|   | XGCB, 55  | XRS, 118  |
|   | XGCB, 56  | XRT, 119  |
|   | XGCP, 57  | XRT, 120  |
|   | XGCR, 58  | XRT, 121  |
|   | XGL, 59   | XRT, 122  |
|   | XGLM, 61  | XRW, 123  |
|   | XGLU, 62  | XS, 124   |
|   | XGM, 64   | XS, 126   |
|   | XGM, 65   | XS, 128   |
|   | XGNP, 66  | XS, 129   |
|   | XGTM, 68  | XS, 130   |
|   | XGUM, 69  | XSSP, 132 |
|   | XISE, 70  | XSTM, 133 |
|   | XKT, 71   | XST, 134  |
|   | XKTM, 72  | Xu, 135   |
|   | XLD, 73   | XSU, 137  |
|   | XLER, 75  | XSW, 138  |
|   | XLFN, 76  | XSW, 139  |
|   | XLKF, 78  | XSUM, 141 |
XTEF, 142
XTLP, 143
XUAD, 145
XUDT, 146
XULF, 147
XULT, 148
XUSP, 149
XUTM, 150
XVEC, 151
XWBF, 152
XWDT, 153
XWFA, 154
XWFP, 155
XWLF, 156
XWSE, 157
XWTM, 158
XZFL, 159

Z

Zero
file, 159