CHAPTER 5

PDOS ASSEMBLY PRIMITIVES

PDOS assembly primitives are assembly language system calls to PDOS. They consist of one word XOP instructions which use XOP vectors 13, 14, and 15. Most calls have error returns, while others return only status or do not return at all.

PDOS calls are divided into four categories: namely, 1) system, 2) console I/O, 3) files, and 4) support primitives.

(CHAPTER 5 PDOS ASSEMBLY PRIMITIVES continued)

5.1 PDOS ASSEMBLY LANGUAGE CALLS

PDOS assembly primitives are one word XOP instructions which use XOP vectors 13, 14, and 15. Most calls have error returns, while others return only status or do not return at all. Calls with error returns continue program execution two bytes beyond the call for a normal return, while an error condition returns immediately after the call instruction. This facilitates an immediate error report primitive or a 'JMP' to an error routine.

```
CALLX  LI R1,FILEN     ;GET FILE NAME
       XSOP            ;OPEN FILE, ERROR?
         JMP ERROR     ;Y
       MOV R1,@SLTN    ;N, SAVE SLOT #
```

PDOS command primitives can be grouped according to the register workspaces they use. Level 0 calls are referred to as subroutines and use your program's workspace for their registers and parameters. These commands are higher level primitives which call disk primitives within PDOS. The call is equivalent to a Branch and Link (BL) instruction.

Level 0 commands:

XAPF,XCHF,XCPY,XGML,XLDF,XLST,XRST
XSZF,XFFN,XBCP,XGLB,XGLM,XGLU,XRDE
XRDN,XTAB,XKTB

Level 1 primitives are for character input and output. These primitives use the level 1 workspace contained in each task control block. Registers R6 through R10 of this workspace are special variables used in console work. None of these primitives have an error return.

Level 1 commands:

XCBC,XGCC,XGCR,XPBC,XPCC,XPCL
XPLC,XPMC,XCLS,XPSC,XIPL

Level 2 primitives are the file manipulation routines. They handle defining, deleting, reading, writing, positioning, locking, and other such file utilities. The level 2 workspace of the task control block is used to transact these commands. Most of these primitives have an error return.

Level 2 commands:

XDFL,XDLF,XROO,XROP,XSOP,XNOP,XCLF
XCFA,XRBF,XRLF,XWBF,XWLF,XPSF,XRWF
XRFA,XWFA,XRNF,XLKF,XULF

Only one task can be executing a level 2 primitive at a time. A lock flag located at >2FE6 is set when a task enters a level 2 primitive and is reset when it returns to the caller. All other tasks making a level 2 call swap while waiting for the flag to be reset.

Level 3 primitives are system subroutines and disk access programs. These include data conversion routines as well as disk read, write, and initialize sector programs contained in the boot area.

Level 3 commands:

XISE,XRSE,XWSE,XRSZ,XGNP,XRTM
XWTM,XRDT,XWDT,XFTD,XCBD,XCBH
XCBM,XCDB,XUDT,XUTM,XLFN,XCTB
XSTM,XRTS

A second lock flag located at >2FE8 is used with the disk programs. This makes these calls autonomous and prevents multiple commands from being sent to the disk controller. It is the responsibility of the disk programs to clear this flag before exiting.

(5.1 PDOS ASSEMBLY LANGUAGE CALLS continued)

Level 4 primitives use the clock workspace. They are for testing and setting events, suspending and locking tasks, and for swapping and returning errors.

Level 4 commands:

XSWP,XSWR,XSER,XERS,XERR,XEXT
XSEF,XSUI,XTEF,XLKT,XULT,XGTM

These primitive levels are summarized as follows:

LEVEL:	LV 0	LV 1	LV 2	LV 3	LV 4
XOP:	XOP 13	XOP 13	XOP 14	XOP 15	XOP 15
WORKSPACE:	BL	L1W	L2W	L3W	CLKWS
	------	------	------	------	------
CALL:	XAPF	XCBC	XDFL*	XISE+	XSWP
	XCHF	XGCC	XDLF*	XRSE+	XSWR
	XCPY	XGCR	XROO*	XWSE+	XSER
	XGML	XPBC	XROP*	XRSZ+	XERS
	XLDF	XPCC	XSOP*	XGNP	XERR
	XLST	XPCL	XNOP*	XRTM	XEXT
	XRST	XPLC	XCLF*	XWTM	XSEF
	XSZF	XPMC	XCFA*	XRDT	XSUI
	XFFN	XCLS	XRBF*	XWDT	XTEF
	XBCP	XPSC	XRLF*	XFTD	XLKT
	XGLB	XIPL	XWBF*	XCBD	XULT
	XGLM		XWLF*	XCBH	
	XGLU		XPSF*	XCBM	
	XRDE		XRHF*	XCDB	
	XRDN		XRFA*	XUDT	
	XTAB		XWFA*	XUTM	
	XKTB		XRNF*	XLFN	
	XFFE		XLKF*	XCTB	
		R6=CNT	XULF*	XSTM	
		R7=PRT		XRTS	
		R8=IMP			
		R10=UNT			

* Level 2 lock
+ Level 3 lock

TMS9900 registers are designated by R0 through R15. Control characters appear as either an up arrow (^) preceding a alphabetic character or as two hexadecimal characters between angle brackets. Special characters such as carriage return, line feed, or escape have special abbreviations in angle brackets.

Registers = R0-R15

```
   ^C = >03
 <LF> = >0A
 <CR> = >0D
<esc> = >1B
```

All calls return to the next word following the XOP, except where an error return is noted in the format. A few special calls also set the status register upon return. Such calls allow the user to select the type of jump required to handle the results.

```
XAPF
 error      <== Error return
....        <== Normal return
```

5.2 SYSTEM CALLS

5.2.1 XCTB - CREATE TASK BLOCK

```
  Mnemonic:     XCTB
     Value:     >2FDD

    Format:     XCTB
                  error

 Registers:  IN  R0  = Task size
                (R1) = Task command line pointer
                 R2  = Task time
                 R3  = I/O port
                 R4  = Optional low memory pointer
                 R5  = Optional high memory pointer

            OUT R0 = Spawned task #
```

```
SETO R0            ;USE CURRENT PAGE
LI R1,FILEN        ;GET FILE NAME
LI R2,1            ;1 TIME PERIOD
CLR R3             ;USE PHANTOM PORT
MOV @>1DC(9),R4    ;GET EUM
MOV R4,R5          ;SET END
AI R4,->0400       ;SET BEGINNING (1K)
XCTB               ;CREATE TASK
  JMP ERROR        ;PROBLEM
MOV R0,TASKN       ;SAVE TASK NUMBER
```

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. Register R0 controls the mode of the new task as well as the task size.

If register R0 is positive, then the first available contiguous memory block equal to R0 (in K bytes) is allocated to the new task. This memory comes from any page or map, but must be contiguous. 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 R0 x 1K bytes. Register R1 points to the new task command line. If R1=0, then the monitor is invoked.

```
If R0>0 then:   R0=Task size
              (R1)=Task command line
                   (0=Monitor)
```

If register R0 is zero, then registers R4 and R5 specify the new task's memory limits in the current map or page. Register R1 specifies the task's starting PC.

```
If R0=0 then:   R1=Program PC
             R4-R5=New task memory limits
                   of current map or page
```

If register R0=-1, then registers R4 and R5 specify the new task's memory limits in the current map or page. Register R1 points to the new task command line. (If R0=0, then the monitor is invoked.)

```
If R0=-1 then: (R1)=Task command line
                    (0=Monitor)
               R4-R5=New task memory limits
                    of current map or page
```

If register R0<-1, then the complement of register R0 specifies the new page, R4 and R5 specify the new task's memory limits, and R1 points to the new task command line.

```
If R0<-1 then: -R0-1=New task memory page
                (R1)=Task command line
                     (0=Monitor)
               R4-R5=New task memory limits
                     of current map or page
```

(5.2.1 XCTB - CREATE TASK BLOCK continued)

The command line is transferred to the spawned program via a system message buffer. The maximum length of a command line is 50 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 -1 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.

Register R2 specifies the number of clock tics the new task executes each time it is scheduled. This value is in 1/125ths of a second but can be changed by the BFIX utility.

R2=Clock tics/time slice

Register R3 specifies the I/O port to be used by the new task. If register R3 is positive, then the port is available for both input and output. If register R3 is negative, then the port is used only for output. If register R3 is zero, then no port is assigned. Only one task may be assigned to any one input port while many tasks may be assigned to an output port. Hence, a port is allocated for input only if it is available. An invalid port assignment does not result in an error.

R3=I/O port

If R3=0, then phantom port (no I/O)

If R3>0, then port is used for I/O

If R3<0, then port is used for output only

Finally, the spawned task's number is returned in register R0 to the parent task. This can be used later to test task status or to kill the task.

Possible Errors:

72 = Too many tasks
73 = Not enough memory

5.2.2 XERR - MONITOR ERROR CALL

Mnemonic: XERR
Value: >2FC4

Format: XERR

Registers: IN R0 = Error code

The MONITOR ERROR CALL primitive returns the task to the PDOS monitor and passes an error code in register R0. PDOS prints 'PDOS ERR', followed by the decimal error number.

```
XRSE                    ;READ SECTOR
  XERR                  ;ERROR
....


LI R0,56                ;RETURN EOF ERROR
XERR
```

Possible Errors: None

5.2.3 XEXT - EXIT TO MONITOR

Mnemonic: XEXT
Value: >2FC5

Format: XEXT
(exits to monitor)

The EXIT TO MONITOR primitive returns a user program to the PDOS monitor. PDOS replies with a <LF>, <CR>, <bell>, and a '.' prompt. The latter two characters are changed by the BFIX utility.

```
XCLF                    ;CLOSE FILE, ERROR?
  XERR                  ;Y, DO ERROR CALL
XEXT                    ;N, RETURN TO MONITOR
```

Possible Errors: None

5.2.4 XFTD - FIX TIME & DATE

Mnemonic: XFTD

Value: >2FD5

Format: XFTD

```
      XFTD          ;GET TIME STAMP
      MOV R0,@TSTP   ;SAVE TIME
      MOV R1,@TSTP+2 ;SAVE DATE
      ....

TSTP  DATA 0,0       ;TIME STAMP SAVE
```

Registers: OUT R0 = (Hours * 256) + Minutes

R1 = ((Year * 16) + Month) * 32 + Day

The FIX TIME & DATE primitive returns a two word encoded time and date generated from the system timers. The resultant codes include month, day, year, hours, and minutes. The ordinal codes can be sorted and used as inputs to the UNPACK DATE and UNPACK TIME routines.

(See 5.2.19 UNPACK DATE and 5.2.21 UNPACK TIME.)

Possible Errors: None

5.2.5 XGML - GET MEMORY LIMITS

Mnemonic: XGML

Value: >2F43

Format: XGML

```
START  XGML        ;GET MEMORY LIMITS
       LI R0,ENDP  ;GET POINTER

START2 CLR *R0+    ;CLEAR MEMORY
       C R0,R1     ;DONE?
         JL START2 ;N
       ....
```

Registers: OUT R0 = Beginning User Storage (BUS)

R1 = End User Memory (EUM)

R9 = Task control block

*Uses registers R0,R1,R9,R11 of calling workspace

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 (>200 beyond register R9) and the end of the user task memory. The task may use up to but not including the upper memory limit.

Register R0 is returned pointing to the beginning of user storage and register R1 to the end of user storage.

Possible Errors: None

5.2.6 XGTM - GET TASK MESSAGE

```
   Mnemonic:    XGTM
      Value:    >2FCB

     Format:    XGTM
                  EQ = Message

  Registers:  IN (R1) = 51 character buffer
             OUT  R0  = Source task #
```

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 register R1, the message buffer is released, and the status is set EQUAL. If no message is found, status is returned NE.

The buffer must to at least 51 bytes in length. Only the first encountered message is returned. Messages are data independent and pass any type of binary data.

```
LOOP    LI R1,BUFFER     ;GET BUFFER
        XGTM             ;LOOK FOR MESSAGE
          JNE NONE
        XPCL             ;MESSAGE, CRLF
        XPLC             ;OUTPUT LINE
        JMP LOOP         ;LOOK AGAIN
*
NONE    ....

BUFFER  BSS 51           ;MESSAGE BUFFER
```

5.2.7 XISE - INIT SECTOR

```
     Mnemonic:     XISE
        Value:     >2FCC

       Format:     XISE
                     error

    Registers:  IN  R0  = Disk #
                    R1  = Logical sector #
                   (R2) = Buffer address
```

The INIT SECTOR primitive is a system-defined, hardware-dependent program which writes 256 bytes of data from a buffer (R2) to a logical sector number (R1) on disk (R0). This routine is meant only to be used for disk initialization and is equivalent to the WRITE SECTOR primitive for all sectors except 0. Sector 0 is not checked for the PDOS ID code.

XISE branches to location >F808 of the boot EPROMs. You may substitute other routines to handle different devices such as high speed disks or bubble memories. The call exits with a INCT R14 and RTWP for a normal return. An error return is made by moving the error number to register R0 of the calling routine (*R13) and doing a RTWP. In either case, the level 3 lock at location >2FE8 must be cleared!

See APPENDIX _ PDOS BOOT:SR.

Possible Errors:

Disk errors

```
        LI R0,DISKN     ;GET DISK #
        CLR R1          ;START AT SECTOR 0
        LI R2,BUFFER    ;GET BUFFER PTR
*
LOOP    XISE            ;WRITE TO DISK
          XERR          ;ERROR
        INC R1          ;MOVE TO NEXT
        CI R1,DISKZ     ;DONE?
          JL LOOP       ;N
        ....



XISE00  ....            ;ROUTINE ENTRY


XISE20  INCT R14        ;NORMAL RETURN
*
XISERT  CLR @>2FE8      ;CLEAR LEVEL 3 LOCK
        RTWP            ;RETURN
*
XISERR  MOV R0,*R13     ;ERROR RETURN
        JMP XISERT      ;RETURN
```

5.2.8 XKTB - KILL TASK BLOCK

```
 Mnemonic:     XKTB
    Value:     >2F50

   Format:     XKTB
                 error
```

Registers: IN R0 = Task #

*Uses registers R0-R3,R9,R11

The KILL TASK BLOCK 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 task number is specified in register R0. If register R0=0, then the current task is killed and its memory deallocated in the system memory bit map.

If R0>0, then the selected task is killed and its memory deallocated. If R0<0, then task number ABS(R0) is killed but its memory is not deallocated in the memory bit map.

The kill process includes releasing the input port assigned to the task, and closing all files associated with the task.

Possible Errors:

74 = No such task
76 = Task locked

```
PREND   SETO R0          ;KILL SELF
        XKTB             ;CALL KILL TASK
          XERR
```

If R0=0, then kill self & deallocate memory

If R0>0, then kill task R0 & deallocate memory

If R0<0, then kill task ABS(R0) & do not deallocate memory

5.2.9 XLKT - LOCK TASK

Mnemonic: XLKT
Value: >2FC9

Format: XLKT

Registers: None

The LOCK TASK primitive locks a task in the run state by setting to nonzero the swap lock variable at memory location >2FEA. This allows only user interrupt routines (not tasks) and the current task to receive CPU cycles. The task remains locked until an UNLOCK TASK (XULT) is executed.

XLKT waits until all locks (Level 2 and Level 3 locks) are cleared before the task is locked.

Possible Errors: None

```
        XLKT            ;LOCK TASK
        SBO 20          ;START CRITICAL PROCESS
*
WAIT    TB -5           ;OK?
          JNE WAIT      ;N
        SBZ 20          ;Y, STOP
        XULT            ;UNLOCK TASK
        ....
```

5.2.10 XRDT - READ DATE

Mnemonic: XRDT
Value: >2FD3

Format: XRDT

Registers: OUT (R1) = MN/DY/YR string

The READ DATE primitive returns the current system date as a nine character string. The format is 'MN/DY/YR' followed by a null. Register R1 points to the string in the monitor work buffer.

Possible Errors: None

```
GETD    XPMC            ;OUTPUT PROMPT
          DATA MES1
        XRDT            ;GET DATE
        XPLC            ;OUTPUT TO SCREEN
        ....

MES1    TEXT 'DATE='
        BYTE 0
```

5.2.11 XRSE - READ SECTOR

```
      Mnemonic:    XRSE
         Value:    >2FCD

        Format:    XRSE
                     error

     Registers:  IN  R0  = Disk #
                     R1  = Sector #
                    (R2) = Buffer pointer
```

The READ SECTOR primitive is a system-defined, hardware-dependent program which reads 256 bytes of data into a memory buffer pointed to by register R2. The disk is selected by register R0. Register R1 specifies the logical sector number to be read.

XRSE branches to location >F800 of the boot EPROMs. You may substitute other routines to handle different devices such as high speed disks or bubble memories. The call exits with a INCT R14 and RTWP for a normal return. An error return is made by moving the error number to register R0 of the calling routine (*R13) and doing a RTWP. In either case, the level 3 lock at location >2FE8 must be cleared!

See APPENDIX _ PDOS BOOT:SR.

Possible Errors:

Disk errors

```
        CLR R0          ;SELECT DISK #0
        CLR R1          ;READ HEADER
        LI R2,BUFFER    ;GET BUFFER
        XRSE            ;READ INTO BUFFER
          XERR          ;ERROR
        ....

BUFFER  BSS 256         ;BUFFER




XRSE00  ....            ;ROUTINE ENTRY


XRSE20  INCT R14        ;NORMAL RETURN
*
XRSERT  CLR @>2FE8      ;CLEAR LEVEL 3 LOCK
        RTWP            ;RETURN
*
XRSERR  MOV R0,*R13     ;ERROR RETURN
        JMP XRSERT      ;RETURN
```

5.2.12 XRTM - READ TIME

```
     Mnemonic:     XRTM
        Value:     >2FD1

       Format:     XRTM

    Registers: OUT (R1) = HR:MN:SC string
```

The READ TIME primitive returns the current time as an nine character string. The format is 'HR:MN:SC' followed by a null. Register R1 points to the string in the monitor work buffer.

Possible Errors: None

```
GETD    XPMC            ;OUTPUT PROMPT
          DATA MES1
        XRTM            ;GET TIME
        XPLC            ;OUTPUT TO SCREEN
        ....

MES1    TEXT 'TIME='
        BYTE 0
```

5.2.13 XRTS - READ TASK STATUS

```
     Mnemonic:     XRTS
        Value:     >2FDF

       Format:     XRTS

    Registers: IN  R0 = Task #
               OUT R1 = Task time
                   LT = Suspended
                   EQ = No task
                   GT = Executing
```

The READ TASK STATUS primitive returns in register R1 and the status register the time parameter of the task specified by register R0. The time reflects the execution mode of the task. If R1 returns zero, then the task is not in the task list. If R1 returns a value greater than zero, then the task is in the run state (executing). If R1 returns a negative value, then the task is suspended pending event -(R1).

The task number is returned from the CREATE TASK BLOCK (XCTB) primitive.

Possible Errors: None

```
        SETO R0         ;USE CURRENT PAGE
        LI R1,FILEN     ;GET FILE NAME
        LI R2,1         ;1 TIME PERIOD
        CLR R3          ;USE PHANTOM PORT
        MOV @>1DC(9),R4 ;GET EUM
        MOV R4,R5       ;SET END
        AI R4,->0400    ;SET BEGINNING (1K)
        XCTB            ;CREATE TASK
          JMP ERROR     ;PROBLEM
*
LOOP    XSWP            ;SWAP WHILE WAITING
        XRTS            ;FOR TASK TO COMPLETE
          JNE LOOP
        NEG R0          ;KILL TASK W/O FREEING
        XKTB            ;MEMORY
          JMP ERROR
```

If R1=0, then not in task list

If R1>0, then task executing

If R1<0, then task suspended on event -R1

5.2.14 XSEF - SET EVENT FLAG

Mnemonic: XSEF
Value: >2FC6

Format: XSEF

Registers: IN R1 = Event

The SET EVENT FLAG primitive sets or resets an event flag bit. The event number is specified in register R1 and is modulo 128. If the content of register R1 is positive, the event bit is set to 1. Otherwise, the bit is reset to 0. A hardware event can be simulated by the XSEF primitive when an event number of 1 through 15 is used.

Events are summarized as follows:

1-15 = Hardware events
16-63 = Software events
64-94 = Software resetting events
95-103 = Input port events
104-111 = Output complete events
112 = 1/5 second event
113 = 1 second event
114 = 10 second event
115 = 20 second event
116 = $TTA active
117 = $LPT active
118-125 = To be assigned
126 = Error message disable
127 = System utility

Possible Errors: None

```
LI R1,30        ;SET EVENT 30
XSEF            ;SET EVENT
....

LI R1,-35       ;RESET EVENT 35
XSEF            ;SET EVENT
....
```

4 types of event flags:

1-15 = Hardware
16-63 = Software
64-94 = Software resetting
95-127 = System

5.2.15 XSTM - SEND TASK MESSAGE

```
   Mnemonic:      XSTM
      Value:      >2FDE

     Format:      XSTM
                    error

  Registers:  IN  R0  = Task #
                 (R1) = Message string
```

The SEND TASK MESSAGE primitive places a 50 character message into the PDOS system message buffer. The message is data independent and is pointed to by register R1.

Register R0 specifies the destination of the message. If register R0 equals -1, and there is no input port (phantom port), then the message is sent to the parent task. Otherwise, register R0 specifies the destination task.

The ability to direct a message to a parent task is very useful in background tasking. An assembler need not know from which task it was spawned and can merely direct any diagnostics to the parent task.

If the destination task number equals -1, the task message is moved to the monitor input buffer and parsed as a command line. This feature is used by the CREATE TASK BLOCK primitive to spawn a new task.

Possible Errors:

78 = Message buffer full

```
ERROR   LI R1,ERRM      ;ERROR, RETURN MESSAGE
        SETO R0         ;  TO PARENT TASK
        XSTM            ;SEND MESSAGE
          XERR          ;PROBLEM
        XEXT            ;DONE
```

R0 = -1 sends message to parent task

5.2.16 XSUI - SUSPEND UNTIL INTERRUPT

```
 Mnemonic:    XSUI
    Value:    >2FC7

   Format:    XSUI

Registers:  IN  R1  = Event
```

```
LI R1,5         ;SUSPEND ON LEVEL 5
XSUI            ;SUSPEND TASK
LI R12,>0180    ;POINT TO AUX PORT
SBO 18          ;ACKNOWLEDGE INTERRUPT
....
```

The SUSPEND UNTIL INTERRUPT primitive suspends the user task until the event specified in register R1 occurs. There are 127 events defined in PDOS. The first 15 (1-15) are hardware events while events 16 through 127 are software events. (Event 0 is ignored.) The event number in register R1 is modulo 128.

A suspended task does not receive any CPU cycles until the event occurs. When the event bit is set, the task begins executing at the next instruction after the XSUI call. The task is immediately scheduled and begins executing for hardware event interrupts. All others are scheduled during the normal swapping functions of PDOS.

A suspended task is indicated in the LIST TASK (LT) command by a minus event number being listed for the task time parameter. When the event occurs, the original time parameter is restored.

```
.LT
 TASK  PAGE  TIME    TB    WS    PC    SR ...

 *0/0  0     3     >42A2 >441C >0654 >D40F ...
  1/0  0     -30   >4AA2 >4A82 >1040 >D00F ...
  2/0  0     -5    >52A2 >5282 >292E >C40F ...
```

Hardware events are enabled by overwriting the appropriate interrupt vector with the workspace and address of the event processor. The interrupt mask bit on the 9901 is set to one, enabling the interrupt. However, you must ensure that the system interrupt mask is high enough to allow the interrupt to occur. Software events are indicated by a single bit being set or reset in an event list.

New interrupt vector
Interrupt enabled at TMS9901

If more than one task is suspended on the same event, only the lowest numbered task is rescheduled for all hardware events. For software events, however, all tasks suspended on the event are rescheduled until the event is reset.

Interrupt disabled at TMS9901

Software event flag bit NOT reset

Once a hardware interrupt occurs, PDOS disables further interrupts on the event level at the system TMS9901 by setting the interrupt mask bit to zero. The system interrupt mask is not affected. Software event flags are not reset and must be processed by the event routine.

Possible Errors: None

5.2.17 XSWP - SWAP TO NEXT TASK

Mnemonic: XSWP
Value: >2FC0

Format: XSWP

```
LOOP    TB 5            ;CONDITION MET?
          JEQ LOOP02    ;Y
        XSWP            ;N, SWAP WHILE WAITING
        JMP LOOP
*
LOOP02  ....
```

The SWAP TO NEXT TASK primitive relinquishes control to the next task in the system task list. This should be used by any routine waiting on I/O or other counters.

Possible Errors: None

5.2.18 XTEF - TEST EVENT FLAG

Mnemonic: XTEF
Value: >2FC8

Format: XTEF

Registers: IN R1 = Event

```
        LI R1,30        ;EVENT 30
        XTEF            ;TEST EVENT FLAG
          JEQ EVENT     ;EVENT = .TRUE.
        ....            ;EVENT = .FALSE.
```

The TEST EVENT FLAG primitive sets the 9900 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 register R1 and is modulo 128. The XTEF primitive is meaningful for software events only (16-127).

Possible Errors: None

5.2.19 XUDT - UNPACK DATE

Mnemonic: XUDT
Value: >2FDA

Format: XUDT

Registers: IN R1 = (Year * 16 + Month) * 32 + Day

OUT (R1) = MN/DY/YR

```
        XFTD          ;FIX TIME & DATE
        XUDT          ;UNPACK DATE
        XPLC          ;PRINT 'MN/DY/YR'
        ....
```

The UNPACK DATE primitive converts a one word encoded date into an eight character string terminated by a null (9 characters). Register R1 contains the encoded date and returns with a pointer to the formatted string. The output of the FIX TIME & DATE routine is valid input to this routine.

(See 5.2.4 FIX TIME & DATE.)

Possible Errors: None

5.2.20 XULT - UNLOCK TASK

Mnemonic: XULT
Value: >2FCA

Format: XULT

```
LOOP    TB 5          ;CONDITION MET?
          JNE LOOP    ;N, WAIT
        SBZ 10        ;Y, RESET
        XULT          ;UNLOCK TASK NOW
```

The UNLOCK TASK primitive unlocks a locked task by clearing the swap lock variable at memory location >2FEA. This allows other tasks to be scheduled and receive CPU time.

(See 5.2.9 XLKT - LOCK TASK.)

Possible Errors: None

5.2.21 XUTM - UNPACK TIME

```
   Mnemonic:     XUTM
      Value:     >2FDB

     Format:     XUTM

  Registers:  IN  R1  = (Hours * 256) + Minutes

             OUT  (R1) = HR:MN
```

```
XFTD            ;GET SYSTEM TIME
MOV R0,R1
XUTM            ;CONVERT TO STRING
XPLC            ;PRINT TIME
....
```

The UNPACK TIME primitive converts a one word encoded date into a 5 character string terminated by a null. Register R1 contains the encoded time and returns with a pointer to the formatted string. The output of the FIX TIME & DATE routine is valid input to this routine.

(See 5.2.4 FIX TIME & DATE.)

Possible Errors: None

5.2.22 XWDT - WRITE DATE

```
   Mnemonic:     XWDT
      Value:     >2FD4

     Format:     XWDT

  Registers:  IN  R0 = Month
                  R1 = Day
                  R2 = Year
```

```
LI R0,12        ;SET DATE TO 12/25/80
LI R1,25
LI R2,80
XWDT            ;SET DATE
....
```

The WRITE DATE primitive sets the system date counters. Register R0 specifies the month and ranges from 1 to 12. Register R1 specifies the day of month and ranges from 1 to 31. Register R2 is the last 2 digits of the year.

Possible Errors: None

5.2.23 XWSE - WRITE SECTOR

Mnemonic:	XWSE
Value:	>2FCE
Format:	XWSE error
Registers:	IN R0 = Disk # R1 = Sector # (R2) = Buffer address

The WRITE SECTOR primitive is a system-defined, hardware-dependent program which writes 256 bytes of data from a buffer, pointed to by register R2, to a logical sector and disk device as specified by registers R1 and R0 respectively.

XWSE branches to location >F804 of the boot EPROMs. You may substitute other routines to handle different devices such as high speed disks or bubble memories. The call exits with a INCT R14 and RTWP for a normal return. An error return is made by passing the error number to register R0 of the calling routine workspace (*R13) and doing a RTWP. In either case, the level 3 lock at location >2FE8 must be cleared upon exit!

See APPENDIX _ PDOS BOOT:SR.

Possible Errors:

Disk errors

```
        CLR R0          ;WRITE TO DISK #0
        LI R1,10        ;WRITE TO SECTOR #10
        LI R2,BUFFER    ;GET BUFFER ADDRESS
        XWSE            ;WRITE
          XERR          ;PROBLEM
        ....

BUFFER  BSS 256         ;DATA BUFFER



XWSE00  ....            ;WRITE SECTOR ENTRY


XWSE20  INCT R14        ;NORMAL RETURN
*
XWSERT  CLR @>2FE8      ;CLEAR LEVEL 3 LOCK
        RTWP            ;RETURN
*
XWSERR  MOV R0,*R13     ;ERROR
        JMP XWSERT      ;RETURN
```

5.2.24 XWTM - WRITE TIME

```
     Mnemonic:     XWTM
        Value:     >2FD2

       Format:     XWTM

    Registers: IN  R0 = Hours
                   R1 = Minutes
                   R2 = Seconds
```

The WRITE TIME primitive sets the system clock time. Register R0 specifies the hour and ranges from 0 to 23. Register R1 specifies the minutes and register R2, the seconds. Both range from 0 to 59.

Possible Errors: None

```
LI R0,23          ;SET TIME TO 23:59:59
LI R1,59
LI R2,59
XWTM              ;SET SYSTEM TIME
```

5.3 CONSOLE I/O PRIMITIVES

5.3.1 XBCP - BAUD CONSOLE PORT

```
Mnemonic:    XBCP
   Value:    >2F49

  Format:    XBCP
               NE = error

Registers: IN R1 = CRU base
              R5 = Console Port #
              R6 = Baud rate

*Uses registers R0,R1,R5,R6,R9,R11,R12
```

The BAUD CONSOLE PORT subroutine initializes any one of the eight PDOS I/O ports and binds a physical TMS9902 UART to a character buffer. The subroutine sets the 9902 character format, receiver and transmitter baud rates, and enables receiver interrupts.

Register R5 selects the console port and ranges from 1 to 8. The system variable ITBCRU, located at address >0096 (>00B6 for 102), points to the input CRU base table. This table binds a physical 9902 UART to a port character buffer and is generated burning PDOS initialization. Entries in this table are changed by the BFIX utility or by a nonzero register R1.

The TMS9902 UART's control register is initialized to 1 start bit, 7 bit character, even parity, and 2 stop bits (11 bits). The receiver and transmitter baud rates are initialized to the same value according to register R6. Register R6 ranges from 0 to 7 or the corresponding baud rates of 19200, 9600, 4800, 2400, 1200, 600, 300, or 110. Either parameter is acceptable.

If R5 is negative, then the associated CRU base address is stored in the UNIT 2 (U2C(9)) variable. The port is bound to any CRU base in register R1.

Interrupts are enabled for input only (SBO 18).

Possible Errors:

64 = Invalid port or baud rate

```
START   LI R1,>320      ;ASSIGN CRU BASE
        LI R5,3         ;  TO PORT 3
        LI R6,19200     ;  WITH 19.2K BAUD
        XBCP            ;BAUD PORT
        ....
```

R5 = Port =	1 =	>0080	TM9900/101MA main port
	2 =	>0180	TM9900/101MA aux port
	3 =	>0E00	ER3232 sel #1 page #0
	4 =	>0A00	ER3232 sel #3 page #0
	5 =	>0A40	ER3232 sel #3 page #1
	6 =	>0A80	ER3232 sel #3 page #2
	7 =	>0AC0	ER3232 sel #3 page #3
	8 =	>0B00	ER3232 sel #3 page #4

R6 = 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

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

5.3.2 XCBC - CHECK FOR BREAK CHARACTER

```
Mnemonic:    XCBC
   Value:    >2F54

  Format:    XCBC
               JL  ^C
               JLT esc
             JEQ   nothing
```

The CHECK FOR BREAK CHARACTER primitive checks the current user input port break flag to see if a break character has been entered. The PDOS break characters are control C (>03) and the escape key (>1B).

A control C sets the break flag positive, while an <escape> character sets the flag negative. The XCBC command samples and clears this flag. The condition of the break flag is returned in the status register.

A LOW condition indicates a ^C has been entered. The break flag and the input buffer are cleared. All subsequent characters entered after the ^C and before the XCBC call are dropped.

A LESS THAN condition indicates an <escape> character has been entered. Only the break flag is cleared and not the input buffer. Thus, the <escape> character remains in the buffer.

The ^C character is interpreted as a hard break and is used to terminate command operations. The <escape> character is a soft break and remains in the input buffer, even though the break flag is cleared by the XCBC command. (This allows an editor to use the escape key for special functions or command termination.)

Possible Errors: None

```
        ....
        XCBC             ;BREAK?
          JL CONTC       ;Y, ^C
          JLT ESCAP      ;Y, ESC
        JMP LOOP         ;N, CONTINUE
*
CONTC   LI R0,'^C'       ;CONTROL C, ECHO '^C'
        XPCC             ;OUTPUT
        JMP BEGIN        ;START AGAIN
*
ESCAP   LI R1,BRKM       ;OUTPUT '>>BREAK'
        XPMC             ;OUTPUT
        XEXT             ;EXIT TO PDOS
*
BRKM    BYTE >0A,>0D     ;BREAK MESSAGE
        TEXT '>>BREAK'
        BYTE 0
```

5.3.3 XCLS - CLEAR SCREEN

Mnemonic: XCLS
Value: >2F5C

Format: XCLS

```
....
XCLS            ;CLEAR SCREEN
XPMC            ;OUTPUT MESSAGE
  DATA MES01
....
```

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 at @>1EA(9). The clear screen variable is initialized from memory location >0090 when the task is created. It is altered after the task is executing by the TERMINAL utility.

The CLEAR SCREEN primitive outputs up to four characters: one or two characters, an escape followed by a character, or an escape, character, escape, and a final character. The one word format allows for two characters. The parity bits cause the escape character to precede each character.

```
CSC(9) = E111 1111 E222 2222
        \\     \ \\     \_____
         \\     \ \\__________ 2nd character
          \\     \ \__________ 2nd escape
           \\     \
            \\     \__________
             \\_______________ 1st character
              \_______________ 1st escape
```

The BFIX utility configures location >0090 for the default codes.

5.3.4 XGCC - GET CONSOLE CHARACTER CONDITIONAL

```
 Mnemonic:    XGCC
    Value:    >2F55

   Format:    XGCC
                EQ => No character
                 L => ^C
                LT => Esc

Registers: OUT R0 = Character*256
```

The GET CONSOLE CHARACTER CONDITIONAL primitive checks the interrupt driven input character buffer and returns the next character in the left byte of register R0. The right byte is cleared.

If the buffer is empty, the EQUAL status bit is set. If the character is a control C (>03), then the break flag and input buffer are cleared, and the status is returned LOW. If the character is the escape character (>1B), then the break flag is cleared and the status is returned LESS THAN.

If no special character is encountered, the character is returned in register R0 and the status set HIGH and GREATER THAN.

If no port has been assigned for input (ie. port 0 or phantom port), then the routine always returns an EQUAL status.

Possible Errors: None

```
        ....
        XGCC            ;CHARACTER?
          JEQ CONT      ;N, CONTINUE
          JL QUIT       ;Y, ^C, QUIT
          JLT NEXT      ;Y, ESC, GOTO NEXT
*
WAIT    XGCR            ;Y, WAIT CHARACTER
        JMP CONT
```

5.3.5 XGCR - GET CONSOLE CHARACTER

```
Mnemonic:     XGCR
   Value:     >2F56

  Format:     XGCR
                 L => ^C
                LT => Esc

Registers: OUT R0 = Character*256
```

```
LOOP    XGCR            ;GET CHARACTER
          JL QUIT       ;^C, DONE
          JLT NEXT      ;CONTINUE
        CI R0,'0'*256   ;NUMBER?
        ....
```

The GET CONSOLE CHARACTER primitive checks for a character from first, the input message pointer (@>18A(9)), second, the assigned input file (@>1E0(9)), and then finally, the interrupt driven input character buffer. If a character is ready, it is returned in the left byte of R0 and the right byte is cleared.

If there is no input message, no assigned console port character, and the interrupt buffer is empty, the task is suspended pending a character interrupt.

The status is returned LOW and the break flag cleared if the returned character is a control C (>03). The input buffer is also cleared. Thus, all characters entered after the ^C and before the XGCR call are dropped.

The status is returned LESS THAN and the break flag cleared if the returned character is the <escape> character (>1B).

For all other characters, the status is returned HIGH and GREATER THAN. The break flag is not affected.

If no port has been assigned for input, (ie. port 0 or phantom port), then the task is suspended indefinitely on event 95.

Possible Errors: None

5.3.6 XGLB - GET LINE IN BUFFER

```
        Mnemonic:     XGLB
           Value:     >2F4A

          Format:     XGLB
                        JLT XXXX  {optional}

       Registers:  IN (R2) = Buffer

                  OUT (R1) = Input string
                      (R9) = Task control block
                        EQ = Carriage return only
                         L = Control C

       *Uses registers R0-R3,R11 of calling workspace
```

The GET LINE IN BUFFER subroutine gets a character line into a buffer pointed to by register R2. A XGCR primitive is used by XGLB and hence characters come from a memory message, a file, or the task console port. The line is delimited by a <CR>. The status returns EQUAL if only a <CR> is entered. Register R1 is returned with a pointer to the first character.

The buffer need only be 80 characters in length since XGLB limits the number of characters to 78. All control characters except <rubout>, <escape>, ^C, and <CR> are ignored.

If an <escape> is entered, the task exits to the PDOS monitor unless a 'JLT' instruction immediately follows the XGLB call. If such is the case, then XGLB returns with status set at 'LT'.

Possible Errors: None

```
OPEN    XPMC            ;PROMPT
          DATA MES01
        LI R2,BUF       ;GET BUFFER ADDRESS
        XGLB            ;GET LINE IN BUFFER
          JLT OPEN      ;DO NOT EXIT ON ESC
          JEQ OPEN10    ;USE DEFAULT
*
OPEN2   XSOP            ;OPEN FILE
          JMP OPEN4     ;ERROR
        ....

OPEN4   CI R0,53        ;'NOT DEFINED' ERROR?
          JNE OPERR     ;N
        XDFL            ;Y, DEFINE FILE
OPERR     XERR          ;ERROR
        JMP OPEN2       ;TRY TO OPEN AGAIN
*
OPEN10  ....

MES01   BYTE >0A,>0D
        TEXT 'FILE='
        BYTE 0
BUF     BSS 80
```

5.3.7 XGLM - GET LINE IN MONITOR BUFFER

```
 Mnemonic:     XGLM
    Value:     >2F4B

   Format:     XGLM
                 JLT XXXX  {optional}

Registers: OUT (R1) = Input string
               (R9) = Task control block
                 EQ = Carriage return only
                  L = Control C

*Uses registers R0-R3,R11 of calling workspace
```

```
OPEN    XGLM            ;GET LINE
        XSOP            ;OPEN FILE
          XEXT          ;ERROR
        ....
```

The GET LINE IN MONITOR BUFFER subroutine gets a character line into the monitor buffer. A XGCR primitive is used by XGLM and hence characters come from a memory message, a file, or the task console port. The line is delimited by a <CR>. The status returns EQUAL if only a <CR> is entered. Register R1 is returned with a pointer to the first character.

The monitor buffer is located 256 bytes into the task control block and is 80 characters in length.

If an <escape> is entered, the task exits to the PDOS monitor unless a 'JLT' instruction immediately follows the XGLB call. If such is the case, then XGLB returns with status set at 'LT'.

Possible Errors: None

5.3.8 XGLU - GET LINE IN USER BUFFER

```
  Mnemonic:     XGLU
     Value:     >2F4C

    Format:     XGLU
                  JLT XXXX  {optional}

 Registers: OUT (R1) = Input string
                (R9) = Task control block
                  EQ = Carriage return only
                   L = Control C

*Uses registers R0-R3,R11 of calling workspace
```

The GET LINE IN USER BUFFER subroutine gets a character line into the user buffer. Register R9 points to the user buffer. A XGCR primitive is used by XGLU and hence characters come from a memory message, a file, or the task console port. The line is delimited by a <CR>. The status returns EQUAL if only a <CR> is entered. Register R1 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 <escape> is entered, the task exits to the PDOS monitor unless a 'JLT' instruction immediately follows the XGLB call. If such is the case, then XGLB returns with status set at 'LT'.

Possible Errors: None

```
GETN    LI R4,DNUM      ;GET DEFAULT #
        XGLU            ;GET LINE
          JEQ GETN2     ;USE DEFAULT
        XCBD            ;CONVERT #
          JLE ERROR
        MOV R1,R4
*
GETN2   MOV R4,SAVE     ;SAVE #
        ....
```

5.3.9 XIPL - INTERRUPT DRIVER PUT LINE

```
     Mnemonic:     XIPL
        Value:     >2F5E

       Format:     XIPL

    Registers:  IN  R0  = Port #
                   (R1) = String
```

The INTERRUPT DRIVER PUT LINE primitive outputs a line to a console port using the transmitter interrupt features of the TMS9902 UART. Register R0 specifies the port number. No check is made as to its range. Register R1 points to the string to be output.

The routine first checks the port output variable and waits until zero. Then, the first character is output, the output variable set, and transmitter empty interrupt enabled. It is the responsibility of the calling program to monitor completion if the line buffer is to be used again. This is done by suspending on the corresponding output event.

The interrupt processor outputs characters until a null character is encountered. When complete, the output variable is cleared and the corresponding output event set.

Possible Errors: None

```
MOV @PRT(9),R0  ;GET CURRENT PORT #
MOV R0,R2
AI R2,103       ;GET CORRESPONDING
MOV R2,R1       ;  OUTPUT EVENT #
NEG R1          ;NEGATE TO RESET
XSEF            ;RESET EVENT
LI R1,MES01     ;GET MESSAGE POINTER
XIPL            ;OUTPUT LINE
MOV R2,R1
XSUI            ;SUSPEND UNTIL DONE
....
```

5.3.10 XPBC - PUT USER BUFFER TO CONSOLE

Mnemonic: XPBC
Value: >2F57

Format: XPBC

Registers: None

The PUT USER BUFFER TO CONSOLE primitive outputs to the user console and/or SPOOL file the ASCII contents of the user buffer. The output string is delimited by the null character. The user buffer is the first 256 bytes of the task control block.

Each character is masked to 7 bits as it is processed. 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 carriage return (>0D) clears the counter. Tabs (>09) are expanded with blanks to MOD 8 character zone fields.

The output routine first sets RTS (SBO 16) and then checks DSR (TB 27) and BUSY (TB 22). If either one is nonzero, PDOS swaps to the next task and waits for both to clear. After the character is output, RTS is reset (SBZ 16).

If UNIT and SPOOL UNIT have coinciding bits, then the processed characters are written to the file slot specified by SPUN (@>1E2(9)). The characters are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

Possible Errors: None

```
CLINE   MOV R9,R2       ;GET USER BUFFER PTR
*
CLINE2
        ....

        MOVB R0,*R2+    ;LOAD BUFFER, DONE?
          JNE CLINE2    ;N
        XPBC            ;Y, OUTPUT BUFFER
        JMP CLINE       ;CONTINUE
```

5.3.11 XPCC - PUT CHARACTER TO CONSOLE

Mnemonic: XPCC
Value: >2F58

Format: XPCC

```
LI R0,'^C'      ;OUTPUT '^C'
XPCC
LI R0,>0A00     ;FOLLOWED BY LF
XPCC
```

Registers: IN R0 = Character

The PUT CHARACTER TO CONSOLE primitive outputs to the user console and/or SPOOL file the ASCII characters in register R0. If only one character is to be output, it is placed in the left byte with the right byte zero. If the right byte is nonzero, it is sent following the left byte.

Each character is masked to 7 bits as it is processed. 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 carriage return (>0D) clears the counter. Tabs (>09) are expanded with blanks to MOD 8 character zone fields.

The output routine first sets RTS (SBO 16) and then checks DSR (TB 27) and BUSY (TB 22). If either one is nonzero, PDOS swaps to the next task and waits for both to clear. After the character is output, RTS is reset (SBZ 16).

If UNIT and SPOOL UNIT have coinciding bits, then the processed characters are written to the file slot specified by SPUN (@>1E2(9)). The characters are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

Possible Errors: None

5.3.12 XPCL - PUT CRLF TO CONSOLE

Mnemonic: XPCL
Value: >2F59

Format: XPCL

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

Registers: None

The PUT CRLF TO CONSOLE primitive outputs to the user console and/or SPOOL file the ASCII characters <LF> and <CR>. The column counter is cleared.

The output routine first sets RTS (SBO 16) and then checks DSR (TB 27) and BUSY (TB 22). If either one is nonzero, PDOS swaps to the next task and waits for both to clear. After the character is output, RTS is reset (SBZ 16).

If UNIT and SPOOL UNIT have coinciding bits, then the processed characters are written to the file slot specified by SPUN (@>1E2(9)). The characters are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

Possible Errors: None

5.3.13 XPLC - PUT LINE TO CONSOLE

Mnemonic: XPLC
Value: >2F5A

Format: XPLC

Registers: IN (R1) = ASCII string

```
        LI R1,MES1      ;OUTPUT MESSAGE
        XPLC
        LI R1,NUMB      ;GET NUMBER
        XCBD            ;CONVERT TO DECIMAL
        XPLC            ;OUTPUT
        ....

NUMB    DATA 0          ;NUMBER HOLDER
MES1    BYTE >0A,>0D    ;MESSAGE #1
        TEXT 'ANSWER='
        BYTE 0
```

The PUT LINE TO CONSOLE primitive outputs to the user console and/or SPOOL file the ASCII character string pointed to by R1. The string is delimited by the null character.

Each character is masked to 7 bits as it is processed. 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 carriage return (>0D) clears the counter. Tabs (>09) are expanded with blanks to MOD 8 character zone fields.

The output routine first sets RTS (SBO 16) and then checks DSR (TB 27) and BUSY (TB 22). If either one is nonzero, PDOS swaps to the next task and waits for both to clear. After the character is output, RTS is reset (SBZ 16).

If UNIT and SPOOL UNIT have coinciding bits, then the processed characters are written to the file slot specified by SPUN (@>1E2(9)). The characters are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

Possible Errors: None

5.3.14 XPMC - PUT MESSAGE TO CONSOLE

```
 Mnemonic:   XPMC
    Value:   >2F5B

   Format:   XPMC
               DATA message
Registers:  None
```

```
        XPMC            ;OUTPUT HEADER
          DATA MES2
        ....

MES2    BYTE >0A,>0D    ;MESSAGE #2
        TEXT 'PDOS REV 2.4'
        BYTE 0
```

The PUT MESSAGE TO CONSOLE command outputs to the user console and/or SPOOL file the ASCII character string pointed to by the word immediately following the PDOS call. The output string is delimited by the null character.

Each character is masked to 7 bits as it is processed. 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 carriage return (>0D) clears the counter. Tabs (>09) are expanded with blanks to MOD 8 character zone fields.

The output routine first sets RTS (SBO 16) and then checks DSR (TB 27) and BUSY (TB 22). If either one is nonzero, PDOS swaps to the next task and waits for both to clear. After the character is output, RTS is reset (SBZ 16).

If UNIT and SPOOL UNIT have coinciding bits, then the processed characters are written to the file slot specified by SPUN (@>1E2(9)). The characters are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

Possible Errors: None

5.3.15 XPSC - POSITION CURSOR

```
 Mnemonic:   XPSC
    Value:   >2F5D

   Format:   XPSC

Registers: IN R1 = x position (Row)
              R2 = y position (Column)
```

The POSITION CURSOR primitive positions the cursor on the console terminal according to the row and column values in registers R1 and R2. Register R1 specifies the row on the terminal and generally ranges from 0 to 23, with 0 being the top row. Register R2 specifies the column of the terminal and ranges from 0 to 79, with 0 being the left-hand column. Register R2 is also loaded into the column counter reflecting the true column of the cursor.

The XPSC primitive outputs either one or two leading characters followed by the row and column. The leading characters output by XPSC are located in PSC (@>1EC(9)) in the task control block. When a task is created, PDOS loads these characters with defaults which come from absolute locations >0092 and >0093.

The row and column characters are biased by >20 is the parity bit of the 1st character is set. Likewise, if the 2nd parity bit is set, then row/column order is reversed. This accommodates must terminal requirements for positioning the cursor.

The BFIX utility is used to change the position cursor codes. The TERMINAL utility changes the codes while the task is executing.

Possible Errors: None

```
OUTM    LI R1,23        ;POSITION TO BOTTOM
        CLR R2          ;  OF SCREEN
        XPSC            ;POSITION
        XPMC            ;OUTPUT MESSAGE
          DATA MES1
        ....
```

5.3.16 XTAB - TAB

```
Mnemonic:     XTAB
   Value:     >2F4F

  Format:     XTAB
                DATA column #
```

Registers: OUT R9 = Task control block

*Uses registers R9,R11 of calling workspace

The TAB subroutine 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 0.

Possible Errors: None

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

5.4 FILE PRIMITIVES

5.4.1 XAPF - APPEND FILE

```
  Mnemonic:      XAPF
     Value:      >2F40

    Format:      XAPF
                   error

Registers:  IN   (R1) = Source file name
                 (R2) = Destination file name

           OUT    R9  = Task control block

*Uses registers R0-R6,R9,R11 of calling workspace
```

The APPEND FILE subroutine is used to append two files together. The source and destination file names are pointed to by registers R1 and R2, respectively. The source file is appended to the end of the destination file. The source file is not altered.

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
62 = File already open
68 = Disk not formatted
69 = No more file slots
Disk errors

```
APFL:   LI R1,SFILEN     ;SOURCE FILE NAME
        LI R2,DFILEN     ;DESTINATION FILE NAME
        XAPF             ;APPEND
          JMP ERROR      ;ERROR RETURN
        ....             ;NORMAL RETURN

SFILEN  TEXT 'FILE1'
        BYTE 0
DFILEN  TEXT 'FILE2'
        BYTE 0
```

5.4.2 XCFA - CLOSE FILE WITH ATTRIBUTES

```
 Mnemonic:     XCFA
    Value:     >2F87

   Format:     XCFA
                 error

Registers:  IN   R1  = FILE ID
                 R2  = File type
```

The CLOSE FILE WITH ATTRIBUTES primitive closes an open file identified by FILE ID. At the same time, the file attributes are updated to the contents of the left byte of register R2. Register R1 contains the FILE ID.

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 is be lost and possibly even the file itself.

Possible Errors:

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

```
        MOV @FILID,R1    ;GET FILE ID
        LI R2,>2000      ;CLOSE AS OBJECT
        XCFA             ;CLOSE FILE
          JMP ERROR
        ....

FILID   DATA 0           ;FILE ID
FILEN   TEXT 'FILENAME:EXT'
        BYTE 0
```

R2 = >8000	AC or Procedure file
= >4000	BN or Binary file
= >2000	OB or 9900 object file
= >1000	SY or Condensed 9900 object file
= >0800	BX or BASIC binary token file
= >0400	EX or BASIC ASCII file
= >0200	TX or Text file
= >0100	Undefined
= >0000	Clear file attributes

FILE ID = (Disk #) x 256 + (File slot index)

5.4.3 XCHF - CHAIN FILE

```
  Mnemonic:     XCHF
     Value:     >2F41

    Format:     XCHF
                error return only

 Registers:  IN  (R1) = File name

 *Uses all registers of calling workspace
```

```
              LI R1,FILEN       ;GET FILE NAME
              XCHF              ;CHAIN FILE
              XERR              ;PROBLEM

      FILEN   TEXT 'NEXTPRGM'
              BYTE 0
```

The CHAIN FILE subroutine is used by the PDOS monitor to execute program files. The primitive chains from one program to another independent of file type.

Register R1 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 9900 object 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. These are located at @>1FA(9), @>1FC(9), and @>1FE(9).

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
61 = No start address
63 = Illegal object tag
64 = Checksum error
65 = Exceeds task size
66 = File not loadable
77 = Procedure not memory resident
Disk errors

5.4.4 XCLF - CLOSE FILE

```
    Mnemonic:     XCLF
       Value:     >2F86

      Format:     XCLF
                    error

   Registers:  IN  R1  = FILE ID
```

The CLOSE FILE primitive closes an open file identified by FILE ID. Register R1 contains the FILE ID. If the file was opened for sequential access and the file was updated, then the END-OF-FILE marker is set at the current file pointer.

If the file was opened for random or shared access, then the END-OF-FILE marker is updated only if the file was extended (ie. data was written after the current END-OF-FILE marker).

If the file has been altered, the current date and time is store in the LAST UPDATE variable of the file directory.

All files must be closed when opened! Otherwise, directory information is lost and possibly even the file itself.

Possible Errors:

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

```
        MOV @FILID,R1    ;GET FILE ID
        XCLF             ;CLOSE FILE
          JMP ERROR
        ....

FILID   DATA 0     ;FILE ID
```

FILE ID = (Disk #) x 256 + (File slot index)

5.4.5 XCPY - COPY FILE

```
 Mnemonic:     XCPY
    Value:     >2F42

   Format:     XCPY
                 error

Registers:  IN R1 = Source file name
               R2 = Destination file name

           OUT R9 = Task control block

*Uses registers R0-R6,R9,R11 of calling workspace
```

```
        LI R1,FILES     ;SOURCE FILE NAME
        LI R2,FILED     ;DESTINATION FILE NAME
        XCPY            ;COPY FILE
          JMP ERROR     ;PROBLEM
        ....            ;CONTINUE

FILES   TEXT 'TEMP'
        BYTE 0
FILED   TEXT 'TEMP:BK/1'
        BYTE 0
```

The COPY FILE primitive copies the source file into the destination file. The source file is pointed to by register R1 and the destination file is pointed to by register R2. A control C halts the copy, prints '^C' to the console, and returns.

The file attributes of the source file are automatically transferred to destination file.

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
62 = File already open
68 = Disk not formatted
69 = No more file slots
70 = Position error
Disk errors

5.4.6 XDFL - DEFINE FILE

```
    Mnemonic:     XDFL
       Value:     >2F80

      Format:     XDFL
                    error

   Registers:  IN  R0  = File size
                  (R1) = File name
```

The DEFINE FILE primitive creates in a PDOS disk directory a new file entry, specified by register R1. A PDOS file name consists of an alpha 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.

Register R0 contains the number of sectors to be initially allocated at file definition. If register R0 is nonzero, then a contiguous file is created with R0 sectors. Otherwise, only one sector is allocated and a non-contiguous tag assigned. 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 past its initial allocation.

Possible Errors:

50 = Invalid file name
51 = File already defined
57 = File directory full
62 = File already open
68 = Disk not formatted
Disk errors

```
CLR R0          ;SEQUENTIAL FILE
LI R1,FILEN1    ;GET FILE NAME
XDFL            ;DEFINE FILE
  XERR          ;ERROR
....
```

```
LI R0,100       ;RANDOM ACCESS FILE
LI R1,FILEN2    ;GET FILE NAME
XDFL            ;DEFINE CONTIGUOUS FILE
  XERR
....
```

R0 > 0 Contiguous file with R0 sectors

R0 = 0 Non-contiguous file

5.4.7 XDLF - DELETE FILE

```
   Mnemonic:    XDLF
      Value:    >2F81

     Format:    XDLF
                  error

  Registers:  IN  (R1) = File name
```

```
        LI R1,FILEN    ;GET FILE NAME PTR
        XDLF           ;DELETE FILE
          JMP ERROR    ;ERROR
        ....           ;NORMAL RETURN

FILEN   TEXT 'TEMP/2'
        BYTE 0
```

The DELETE FILE primitive removes from the disk directory the file whose name is pointed to by register R1 and releases all sectors associated with that file for use by other files on that same disk. A file cannot be deleted if it is delete (*) or write (**) protected.

Possible Errors:

```
        50 = Invalid file name
        53 = File not defined
        58 = File delete or write protected
        62 = File already open
        68 = Disk not formatted
        Disk errors
```

5.4.8 XFFN - FIX FILE NAME

```
     Mnemonic:    XFFN
        Value:    >2F48

       Format:    XFFN
                    error

    Registers:  IN (R1) = File name

               OUT  R0  = Disk #
                   (R1) = Fixed file name
                    R9  = Task control block

    *Uses registers R0-R3,R9,R11 of calling workspace.
```

```
XGLU            ;GET INPUT LINE
XFFN            ;FIX FILE NAME
  XERR          ;ERROR IN NAME
....
```

The FIX FILE NAME subroutine parses a character string for file name, extension, directory level, and disk number. The results are returned in the 32 character monitor work buffer (MWB(9)). Register R0 is also returned with the disk number. The error return is used for an invalid file name.

The monitor work buffer is cleared and the following assignments are made:

```
     @0(1) = File name
     @8(1) = File extension
    @11(1) = File directory level
```

```
             0   2   4   6   8   10  12  14  16
             '---'---'---'---'---'---'---'---'...
(R1) ==>     | File name     | Ext |L| 00==>
             '---------------'-----'-'-------  ...
```

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

Possible Errors:

    50 = Invalid file name

5.4.9 XLDF - LOAD FILE

```
   Mnemonic:     XLDF
      Value:     >2F44

     Format:     XLDF
                   error

  Registers:  IN  R0  = Start memory address
                  R1  = End memory address
                 (R2) = File name

             OUT  R0  = Entry address
                  R9  = Task control block
```

*Uses all registers except R10

The LOAD FILE primitive reads and loads TI9900 object code into user memory. The file name pointer is passed in register R2. Registers R0 and R1 specify the memory bounds for the relocatable load. The file must be typed 'OB' or 'SY'.

The TI9900 object must be relocatable and register R0 is returned to the calling routine with the program entry address. If register R0 equals zero, no start has been found. Valid TI9900 object tags for 'OB' files are defined as follows:

Tag	Meaning	Tag	Meaning
0	= Program ID	8	= Ignore checksum
1	= Illegal	9	= Illegal
*2	= Relocatable entry	*A	= Relocatable address
3	= Illegal	*B	= Absolute data
4	= Illegal	*C	= Relocatable data
5	= Illegal	D	= Illegal
6	= Illegal	E	= Illegal
7	= Checksum	F	= End of record

A 'SY' file is generated from an 'OB' file by the SYFILE utility. The condensed object code contains only 4 types of object tags, each followed by a 2-byte binary number. These are indicated by an asterisk (*) in the above table.

Possible Errors:

63 = Illegal tag character
64 = Checksum error
65 = Memory limit exceeded
66 = File not loadable
Disk errors

```
XGML           ;GET MEMORY LIMITS
AI R0,>0100    ;ADD DISPLACEMENT
LI R2,FILEN    ;GET FILE NAME
XLDF           ;LOAD FILE
  XERR         ;ERROR
MOV R0,R0      ;OK ADDRESS?
  JEQ ERROR    ;N
B *R0          ;Y, GOTO ROUTINE
```

```
00000IDT=HEREA0000B6865B6C6CC6F5F20000F
```

```
AxxBheB11Co_2xx
```

5.4.10 XLFN - LOOKUP FILE NAME

```
 Mnemonic:      XLFN
    Value:      >2FD8

   Format:      XLFN
                  Found
                Not found

Registers:  IN  R0  = Disk #
               (R1) = File name

           OUT  R3  = FILE ID
                R7  = File slot address
```

```
XNOP    MOV @2(13),R1   ;GET FILE ID
        XFNM            ;FIX FILE NAME
          XSER          ;ERROR
        XLFN            ;LOOKUP NAME, FOUND?
          JMP ERR62     ;Y, FILE ALREADY OPEN
        ....

ERR62   XERS            ;FILE ALREADY OPEN
          DATA 62
```

The LOOKUP FILE NAME primitive searches through the file slot table for the file name as specified by registers R0 and R1. If the name is not found, register R3 returns with a -1. Otherwise, register R3 returns the associated FILE ID and register R7 the address of the file slot.

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

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

Possible Errors: None

5.4.11 XLKF - LOCK FILE

```
   Mnemonic:   XLKF
      Value:   >2F91

     Format:   XLKF
                 error

  Registers:  IN R1 = FILE ID
```

```
MOV @FILEID,R1  ;GET FILE ID
XLKF            ;LOCK FILE
  JMP ERROR     ;PROBLEM
....
```

The LOCK FILE primitive locks an OPENed file such that no other task can gain access until an UNLOCK FILE (XULF) is executed.

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). Only the locking task has access to the locked file.

Possible Errors:

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

5.4.12 XLST - LIST FILE DIRECTORY

```
   Mnemonic:    XLST
      Value:    >2F45

     Format:    XLST
                  error

  Registers:  IN (R1) = List string

             OUT  R9  = Task control block

  *Uses registers R0-R8,R9,R11
```

```
MLST    XGNP            ;GET SELECT LIST
          JH MLST02     ;PARAMETER OK
        LI R1,NULL      ;USE NULL STRING
*
MLST02  XLST            ;CALL FOR LIST
          XERR          ;ERROR
        XEXT            ;EXIT TO MONITOR
```

The LIST FILE DIRECTORY subroutine causes PDOS to output to the console terminal a formatted file directory listing, according to the select string pointed to by register R1. The output is interrupted at any time by a character being entered on the console port. An <esc> character returns control to the PDOS monitor.

(See 4.17 LIST DIRECTORY.)

Possible Errors: Disk Errors

5.4.13 XNOP - OPEN SHARED RANDOM FILE

```
 Mnemonic:    XNOP
    Value:    >2F85

   Format:    XNOP
                error

Registers:  IN  (R1) = File name

            OUT  R0  = File type
                 R1  = FILE ID
```

```
         LI R1,FILEN      ;GET FILE NAME
         XNOP             ;OPEN SHARED FILE
           JMP ERROR      ;ERROR
         MOV R0,@FILET    ;SAVE TYPE
         MOV R1,@FILID    ;SAVE FILE ID
         ....

FILET    DATA 0
FILID    DATA 0
FILEN    TEXT 'FILENAME:EXT'
         BYTE 0
```

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. A FILE ID and file type are returned to the calling program in registers R1 and R0, 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 R1) is a 2-byte number. The left byte is the disk number and the right byte is the channel buffer index. The file type is returned in register R0.

FILE ID = (Disk #) x 256 + (File slot index)

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

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

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
62 = File already open
68 = Disk not formatted
69 = No more file slots
Disk errors

5.4.14 XPSF - POSITION FILE

```
 Mnemonic:     XPSF
    Value:     >2F8C

   Format:     XPSF
                 error

Registers:  IN    R1 = FILE ID
                R2,R3 = Byte position
```

The POSITION FILE primitive moves the file byte pointer to any byte position within a file. The FILE ID is given in register R1 and the two word byte index is specified in registers R2 and R3.

The file must have been opened for random access (ROPEN or SOPEN). An error occurs if the byte index is greater than the current End-of-File marker.

A contiguous file greatly enhances the speed of the position command since the desired sector is directly computed. However, the position command 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 alters the file type to non-contiguous and random access is much slower.

Possible Errors:

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

```
        MOV @FILID,R1   ;GET FILE ID
        MOV @RECN,R2    ;GET RECORD #
        MPY @C36,R2     ;GET BYTE INDEX
        XPSF            ;POSITION WITHIN FILE
          XERR
        ....

FILID   DATA 0          ;FILE ID
RECN    DATA 0          ;RECORD #
C36     DATA 36         ;BYTES/RECORD
```

5.4.15 XRBF - READ BLOCK

```
    Mnemonic:     XRBF
       Value:     >2F88

      Format:     XRBF
                    error

   Registers:  IN   R0  = # of bytes to be read
                    R1  = FILE ID
                   (R2) = Buffer address

              OUT   R3  = # of bytes read on error
```

The READ BLOCK primitive reads the number of bytes specified in register R0 from the file specified by the FILE ID in register R1 into the user memory as pointed to by register R2. 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 R0 and the number of bytes actually read in register R3.

The read is independent of the data content. The buffer pointer in register R2 is on any byte boundary. The buffer is not terminated with a null.

A byte count of zero in register R0 results in one byte being read from the file. This facilitates single byte data acquisition.

Possible Errors:

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

```
         LI R0,NUMB       ;GET NUMBER OF BYTES
         MOV @FILID,R1    ;GET FILE ID
         LI R2,BUFF       ;GET BUFFER POINTER
         XRBF             ;READ DATA
           JMP ERROR
         ....

ERROR    CI R0,56         ;EOF?
           JNE ERROR2     ;N
         MOV R3,@NUMB     ;Y, SAVE # BYTES READ
         ....

FILID    DATA 0
NUMB     DATA 0           ;# OF BYTES TO READ
BUFF     BSS 132          ;BUFFER
```

```
         CLR R0           ;READ 1 CHARACTER
         MOV @FILED,R1    ;GET FILE SLOT ID
         STWP R2          ;READ CHARACTER INTO R0
         XRBF             ;READ CHARACTER
           JMP ERROR
         ....
```

5.4.16 XRDE - READ DIRECTORY ENTRY

```
 Mnemonic:     XRDE
    Value:     >2F4D

   Format:     XRDE
                 error

Registers:  IN  R0  = Disk #
                R1  = Read flag
               (R2) = Last 32 byte directory entry
           @>1F2(9) = Sector #
           @>1F4(9) = # of directory entries

           OUT  R0  = Disk #
               (R2) = Next 32 byte directory entry
                R9  = Task control block
           @>1F2(9) = Sector #
           @>1F4(9) = # of directory entries

*Uses registers R0-R4,R9,R11 of calling workspace
```

The READ DIRECTORY ENTRY subroutine reads sequentially through a disk directory. If register R1 is zero, then the routine begins with the first directory entry. If register R1 is nonzero, then based on the last directory entry (pointed to by register R2), the next entry is read.

The calling routine must maintain registers R0 and R2, the user I/O buffer, and temps >1F2(9) and >1F4(9) of the task control block between calls to XRDE.

Possible Errors:

53 = File not defined (End of directory)
68 = Disk not formatted
Disk errors

```
START   CLR R1            ;BEGIN WITH 1ST ENTRY
        JMP LOOP02
*
LOOP    SETO R1           ;READ NEXT ENTRY
*
LOOP02  MOV @TSM1(9),R0   ;GET DISK #
        XRDE              ;READ DIRECTORY ENTRY
          XERR            ;ERROR
        MOV @12(2),R4     ;GET FILE TYPE
        ....
```

5.4.17 XRDN - READ DIRECTORY NAME

```
     Mnemonic:     XRDN
        Value:     >2F4E

       Format:     XRDN
                     error

    Registers:  IN  R0   = Disk #
                   MWB   = File name

               OUT  R0   = Disk #
                    R1   = Sector # in memory
                   (R2)  = Directory entry
                    R9   = Task control block
```

*Uses registers R0-R5,R9,R11 of calling workspace

The READ DIRECTORY NAME subroutine reads directory entries by file name. Register R0 specifies the disk number. The file name is located in the Monitor Work Buffer (MWB) in a fixed format. Several other parameters are returned in the monitor TEMP storage of the user status buffer. These variables assist in the housekeeping operations on the disk directory.

(See 5.4.8 FIX FILE NAME.)

Possible Errors:

53 = File not defined
68 = Disk not formatted
Disk errors

```
OPENF   MOV @2(13),R1    ;GET FILE NAME POINTER
        XFNM             ;FIX NAME IN MWB
          XSER           ;ERROR
        XRDN             ;READ DIRECTORY ENTRY
          XSER           ;ERROR
        CB *R2,@B24      ;$?  (DRIVER?)
        ....
```

@>172(9) => Monitor Work Buffer

5.4.18 XRFA - READ FILE ATTRIBUTES

Mnemonic: XRFA
Value: >2F8E

Format: XRFA
error

Registers: IN (R1) = File name

OUT R2 = File attribute

The READ FILE ATTRIBUTES primitive returns in register R2 the 16-bit file attributes word. The file name is pointed to by register R1. File attributes are defined as follows:

>80xx	AC	- PROCEDURE FILE
>40xx	BN	- BINARY FILE
>20xx	OB	- 9900 OBJECT FILE
>10xx	SY	- SYSTEM OBJECT FILE
>08xx	BX	- BASIC TOKEN FILE
>04xx	EX	- BASIC ASCII SOURCE FILE
>02xx	TX	- ASCII TEXT FILE
>01xx	UD	- USER DEFINED FILE
>xx04	C	- CONTIGUOUS FILE
>xx02	*	- DELETE PROTECT
>xx01	**	- DELETE AND WRITE PROTECT

Possible Errors:

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

```
       LI R1,FILEN     ;GET FILE NAME
       XRFA            ;READ FILE ATTRIBUTES
         XERR          ;PROBLEM
       SRL R2,2        ;BINARY FILE?
         JNC PNO       ;N
       ....            ;Y

FILEN  TEXT 'PRGM:BIN'
       BYTE 0
```

5.4.19 XRLF - READ LINE

```
 Mnemonic:     XRLF
    Value:     >2F89

   Format:     XRLF
                 error

Registers:  IN  R1  = FILE ID
               (R2) = Buffer address

           OUT  R0  = Error #
                R3  = # of bytes read on error
```

```
        MOV @FILID,R1    ;GET FILE ID
        LI R2,BUFF       ;GET BUFFER POINTER
        XRLF             ;READ LINE
          JMP ERROR
        ....

FILID   DATA 0
BUFF    BSS 132  ;MAXIMUM BUFFER NEEDED
```

The READ LINE primitive reads one line, delimited by a carriage return <CR>, from the file specified by the FILE ID in register R1. If a <CR> is not encountered after 132 characters, then the line and primitive are terminated. Register R2 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 R0 and the number of bytes actually read in register R3.

The line read is dependent upon the data content. All line feeds <LF> are dropped from the data stream and the <CR> is replaced with a null. The buffer pointer in register R2 is on any byte boundary. The buffer is not terminated with a null on an error return.

Possible Errors:

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

5.4.20 XRNF - RENAME FILE

```
     Mnemonic:     XRNF
        Value:     >2F90

       Format:     XRNF
                     error

    Registers:  IN (R1) = Old file name
                   (R2) = New file name
```

The RENAME FILE primitive renames a file in a PDOS disk directory. The old file name is pointed to by register R1. The new file name is pointed to by register R2.

The XRNF command 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.

```
        LI R1,FILEN1     ;GET OLD FILE NAME
        LI R2,FILEN2     ;GET NEW FILE NAME
        XRNF             ;RENAME FILE
          XERR           ;PROBLEM
        LI R2,LEVEL      ;GET NEW LEVEL
        XRNF             ;CHANGE DIRECTORY LEVEL
          XERR
        ....

LEVEL   DATA 10
FILEN1  TEXT 'OBJECT:OLD'
        BYTE 0
FILEN2  TEXT 'OBJECT:NEW'
        BYTE 0
```

Possible Errors:

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

5.4.21 XROO - OPEN READ ONLY RANDOM FILE

```
 Mnemonic:      XROO
    Value:      >2F82

   Format:      XROO
                  error

Registers:  IN  (R1) = File name

            OUT  R0  = File type
                 R1  = FILE ID
```

The OPEN READ ONLY 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 type 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 type is returned in register R0.

Since the file cannot be altered, it cannot be extended nor is the LAST UPDATE parameter changed when it is closed. All data transfers are buffered through a channel buffer and data movement to and from the disk is by full sectors.

A new file slot is allocated for each XROO call even if the file is already open. The file slot is allocated beginning with slot 1 to 32.

Possible Errors:

50 = Invalid file name
53 = File not defined
62 = File already open
68 = Disk not formatted
69 = No more file slots
Disk errors

```
        LI R1,FILEN     ;GET FILE NAME
        XROO            ;OPEN READ ONLY FILE
          JMP ERROR     ;ERROR
        MOV R0,@FILET   ;SAVE TYPE
        MOV R1,@FILID   ;SAVE FILE ID
        ....

FILET   DATA 0
FILID   DATA 0
FILEN   TEXT 'FILENAME:EXT'
        BYTE 0
```

FILE ID = (Disk #) x 256 + (File slot index)

5.4.22 XROP - OPEN RANDOM FILE

```
 Mnemonic:     XROP
    Value:     >2F83

   Format:     XROP
                 error

Registers:  IN  (R1) = File name

            OUT  R0  = File type
                 R1  = FILE ID
```

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 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 R1) is a 2-byte number. The left byte is the disk number and the right byte is the channel buffer index. The file type is returned in register R0.

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.

Possible Errors:

50 = Invalid file name
53 = File not defined
62 = File already open
68 = Disk not formatted
69 = No more file slots
Disk errors

```
        LI R1,FILEN      ;GET FILE NAME
        XROP             ;OPEN RANDOM FILE
          JMP ERROR      ;ERROR
        MOV R0,@FILET    ;SAVE TYPE
        MOV R1,@FILID    ;SAVE FILE ID
        ....

FILET   DATA 0
FILID   DATA 0
FILEN   TEXT 'FILENAME:EXT'
        BYTE 0
```

FILE ID = (Disk #) x 256 + (File slot index)

5.4.23 XRST - RESET FILES

Mnemonic: XRST
Value: >2F46

Format: XRST

Registers: IN R1 = Reset type

```
DONE    SETO R1         ;CLOSE ALL TASK FILES
        XRST
        ....

        MOV @DISKN,R1   ;PREPARE TO REMOVE DISK
        XRST            ;CLOSE ALL FILES
        ....            ;REMOVE DISK
```

The RESET FILES primitive closes all open files either by task or disk number. The command also clears the assigned input FILE ID. If register R1 equals -1, then all files associated with the current task are closed. Otherwise, register R1 specifies a disk and all files opened on that disk are closed.

XRST has no error return and hence closes all files even though errors occur in the close process. This is necessary since files may be opened on a write protected disk, for instance, and a error occurs before the files could be closed.

Possible Errors: None

5.4.24 XRWF - REWIND FILE

Mnemonic: XRWF
Value: >2F8D

Format: XRWF
  error

Registers: IN R1 = FILE ID

```
REWIND  MOV @FILID,R1   ;GET FILE ID
        XRWF            ;REWIND FILE
          XERR          ;PROBLEM
        ....

FILID   DATA 0
```

The REWIND FILE primitive positions the file specified by the FILE ID in register R1, to byte position zero.

Possible Errors:

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

5.4.25 XSOP - OPEN SEQUENTIAL FILE

```
  Mnemonic:     XSOP
     Value:     >2F84

    Format:     XSOP
                  error

 Registers:  IN  (R1) = File name

             OUT  R0  = File type
                  R1  = FILE ID
```

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 R1) is a 2-byte number. The left byte is the disk number and the right byte is the channel buffer index. The file type is returned in R0.

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

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

Possible Errors:

- 50 = Invalid file name
- 53 = File not defined
- 62 = File already open
- 68 = Disk not formatted
- 69 = No more file slots
- Disk errors

```
        LI R1,FILEN      ;GET FILE NAME
        XSOP             ;OPEN SEQUENTIAL FILE
          JMP ERROR      ;ERROR
        MOV R0,@FILET    ;SAVE TYPE
        MOV R1,@FILID    ;SAVE FILE ID
        ....

FILET   DATA 0
FILID   DATA 0
FILEN   TEXT 'FILENAME:EXT'
        BYTE 0
```

FILE ID = (Disk #) x 256 + (File slot index)

5.4.26 XSZF - SIZE DISK

```
  Mnemonic:     XSZF
     Value:     >2F47

    Format:     XSZF
                  error

 Registers:  IN R0 = Disk #

            OUT R5 = Largest contiguous block
                R6 = Number of sectors allotted
                R7 = Number of sectors used
                R8 = Number of free sectors
                R9 = Task control block

*Uses registers R1-R8,R9,R11 of calling workspace
```

The SIZE DISK subroutine returns disk size parameters in registers R5, R6, R7, and R8. Register R7 returns the total number of sectors used by all files. Register R6 returns the number of sectors allocated for file storage.

Register R8 is calculated from the disk sector bit map and reflects the number of sectors available for file allocation. Register R5 is returned with the size of the largest block of contiguous sectors. This is useful in defining large files.

Possible Errors:

68 = Disk not formatted
Disk errors

```
        CLR R0          ;SELECT DISK #0
        XSZF            ;GET DISK SIZE
          XERR          ;ERROR
        MOV R8,R1
        XCBM            ;OUTPUT FREE
          DATA SPM1
        XPLC            ;PRINT
        MOV R5,R1
        XCBM            ;OUTPUT LARGEST
          DATA SPM2     ;  CONTIGUOUS BLOCK
        XPLC
        XTAB            ;TAB TO COLUMN 20
          DATA 20
        MOV R7,R1
        XCBM            ;OUTPUT USED
          DATA SPM3
        XPLC            ;PRINT
        MOV R6,R1
        XCBM            ;OUTPUT ALLOCATED
          DATA SPM4
        XPLC            ;PRINT
        XEXT
*
SPM1    BYTE >0A,>0D
        TEXT 'FREE:'
        BYTE 0
SPM2    BYTE >2C,0
SPM3    TEXT 'USED:'
        BYTE 0
SPM4    TEXT '/'
        BYTE 0
```

5.4.27 XULF - UNLOCK FILE

```
    Mnemonic:     XULF
       Value:     >2F92

      Format:     XULF
                    error

   Registers:  IN R1 = FILE ID
```

The UNLOCK FILE primitive unlocks a locked file for access by any other task.

(See 5.4.11 XLKF - LOCK FILE.)

Possible Errors:

```
        52 = File not open
        59 = Invalid file slot
        Disk errors
```

```
        MOV @FILID,R1   ;GET FILE ID
        XULF            ;UNLOCK FILE
          XERR
        ....

FILID   DATA 0          ;FILE ID
```

5.4.28 XWBF - WRITE BLOCK

```
Mnemonic:     XWBF
   Value:     >2F8A

  Format:     XWBF
               error

Registers:  IN R0  = Byte count
               R1  = FILE ID
              (R2) = Buffer address
```

```
        LI R0,252       ;WRITE FULL SECTOR
        MOV @FILID,R1   ;GET ID
        LI R2,BUFFER    ;GET BUFFER ADDRESS
        XWBF            ;WRITE TO FILE
          XERR
        ....

FILID   DATA 0          ;FILE ID
BUFFER  BSS 252         ;SECTOR BUFFER
```

The WRITE BLOCK primitive writes from a memory buffer, pointed to by register R2, to a disk file specified by the FILE ID in register R1. Register R0 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 R2 is on any byte boundary. The write operation is not terminated with a null.

A byte count of zero in register R0 results in no data being written to the file.

R0 = 0 Write no data

If it is necessary for the file to be extended, PDOS first uses sectors already linked to the file. If a null or end link is found, a new sector obtained from the disk sector bit map is linked to the end of the file. If the file was contiguous, it is retyped as a non-contiguous file.

Extended file

Contiguous changes to non-contiguous

Possible Errors:

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

5.4.29 XWFA - WRITE FILE ATTRIBUTES

```
 Mnemonic:     XWFA
    Value:     >2F8F

   Format:     XWFA
                 error

Registers:  IN (R1) = File name
               (R2) = ASCII file attributes
```

The WRITE FILE ATTRIBUTES primitive sets the attributes of the file specified by the file name pointed to by register R1. Register R2 points to an ASCII string containing the new file attributes. The format is:

```
(R2) = {file type}{protection}

     {file type} = AC - PROCEDURE FILE
                   BN - BINARY FILE
                   OB - 9900 OBJECT
                   SY - SYSTEM FILE
                   BX - BASIC TOKEN FILE
                   EX - BASIC SOURCE FILE
                   TX - TEXT FILE
                   UD - USER DEFINED FILE

    {protection} = *  - Delete protect
                   ** - Delete and Write protect
```

If register R2 equals zero, then all flags, with the exception of the contiguous flag, are cleared. If register R2 points to a '#', then the contiguous flag is cleared.

Possible Errors:

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

```
        LI R1,FILEN      ;GET FILE NAME
        LI R2,CLRC       ;CLEAR CONTIGUOUS
        XWFA             ;WRITE ATTRIBUTE
          XERR
        LI R2,PROTF      ;SET BINARY & PROTECTED
        XWFA             ;SET
          XERR
        ....

FILEN   TEXT 'DATA:BIN'
        BYTE 0
CLRC    TEXT '#'
        BYTE 0
PROTF   TEXT 'BN**'
        BYTE 0
```

5.4.30 XWLF - WRITE LINE

```
  Mnemonic:     XWLF
     Value:     >2F8B

    Format:     XWLF
                  error

Registers:  IN  R1  = FILE ID
               (R2) = Buffer address
```

```
        MOV @FILID,R1   ;GET FILE ID
        LI R2,LINE      ;GET LINE
        XWLF            ;WRITE LINE
          XERR          ;ERROR
        ....

FILID   DATA 0          ;FILE ID
LINE    BYTE >0A,>0D
        TEXT 'NO DIAGNOSTICS'
        BYTE 0
```

The WRITE LINE primitive writes a line delimited by a null character to the disk file specified by the FILE ID in register R1. Register R2 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.

Null delimiter

The write line command is independent of the data content, with the exception that a null character terminates the string. The buffer pointer in register R2 is on any byte boundary. A single write operation continues until a null character is found.

Extended file

Contiguous changes to non-contiguous

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 the file was contiguous, it is retyped as a non-contiguous file.

Possible Errors:

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

5.5 SUPPORT PRIMITIVES

5.5.1 XCBD - CONVERT BINARY TO DECIMAL

Mnemonic: XCBD
Value: >2FD6

Format: XCBD

Registers: IN R1 = number

OUT (R1) = string pointer

```
        MOV @NUMB,R1    ;GET NUMBER
        XCBD            ;CONVERT TO PRINT
        MOV R1,@SAVE    ;SAVE POINTER
        XPLC            ;PRINT
        ....
*
NUMB    DATA 1234       ;NUMBER HOLDER
SAVE    DATA 0          ;SAVE POINTER
```

The CONVERT BINARY TO DECIMAL primitive converts a 16 bit, 2's complement number to a character string. The number to be converted is passed to XCBD in register R1. Register R1 is also returned with a pointer to the converted character string located in the monitor work buffer. Leading zeros are suppressed and a negative sign is the first character for negative numbers. The string is delimited by a null.

Possible Errors: None

5.5.2 XCBH - CONVERT BINARY TO HEX

Mnemonic: XCBH
Value: >2FD7

Format: XCBH

Registers: IN R1 = number

OUT (R1) = string pointer

```
        MOV @NUMB,R1    ;GET NUMBER
        XCBH            ;GET HEX CONVERSION
        MOV R1,@SAVE    ;SAVE POINTER
        LI R0,' >'      ;ADD HEX SIGN
        XPCC            ;PRINT
        XPLC            ;PRINT 4 HEX CHARACTERS
        ....
*
NUMB    DATA 1234       ;NUMBER HOLDER
SAVE    DATA 0          ;SAVE POINTER
```

The CONVERT BINARY TO HEX primitive converts a 16-bit number to its hexadecimal (base 16) representation. The number is passed in register R1 and a pointer to the ASCII string is also returned in register R1. The converted string is in the monitor work buffer and consists of four hexadecimal characters followed by a null.

Possible Errors: None

5.5.3 XCBM - CONVERT UNSIGNED BINARY TO DECIMAL W/MESSAGE

```
  Mnemonic:   XCBM
     Value:   >2FD8

    Format:   XCBM
                DATA message

Registers:  IN   R1  = number

           OUT  (R1) = string pointer
```

The CONVERT UNSIGNED BINARY TO DECIMAL W/MESSAGE primitive converts a 16 bit, unsigned number to a character string. The output string is preceded by the string whose address immediately follows the call. The string can be up to 24 characters in length and is terminated by a null character. The number to be converted is passed to XCBM in register R1. Register R1 is also returned with a pointer to the converted character string located in the monitor work buffer. Leading zeros are suppressed and the result ranges from 0 to 65535.

Possible Errors: None

```
        MOV @NUMB,R1    ;GET NUMBER
        XCBM            ;CONVERT TO PRINT
          DATA MES1
        MOV R1,@SAVE    ;SAVE POINTER
        XPLC            ;PRINT
        ....
*
NUMB    DATA 1234       ;NUMBER HOLDER
SAVE    DATA 0          ;SAVE POINTER
MES1    BYTE >0A,>0D
        TEXT 'NUMB='
        BYTE 0
```

5.5.4 XCDB - CONVERT DECIMAL TO BINARY

```
Mnemonic:     XCDB
Value:        >2FD9

   Format:    XCDB
                JL  no number
                JH  number
                JEQ number w/o null delimiter

Registers:  IN  (R1) = string pointer

          OUT    R0  = delimiter
                 R1  = number
                (R2) = updated string pointer
```

The CONVERT DECIMAL TO BINARY primitive converts an ASCII string of characters to a 16 bit, 2's complement number. The result is returned in register R1 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.

A LOW status indicates that no conversion was possible. Register R0 is returned with the first character and register R2 points immediately after it.

A HIGH status indicates that a good conversion has been made, and the result is found in register R1. Register R2 is returned with an updated pointer and register R0 is set to zero.

A EQUAL status indicates that a conversion was made but the ASCII string was not terminated with a null character. The result is returned in register R1 and the non-numeric, non-null character is returned in register R0. Register R2 has the address of the next character.

Possible Errors: None

```
         MOV @PTR,R1       ;GET STRING POINTER
         MOV @DFP2,R3      ;GET 2ND DEFAULT
         XCDB              ;CONVERT
           JL ERROR        ;NO NUMBER
           JH CONT         ;OK
         CI R0,>2C00       ;COMMA DELIMITER?
           JNE ERROR       ;N, ERROR
         MOV R2,R1         ;Y, GET NEXT NUMBER
         MOV R1,R3         ;SAVE FIRST RESULT
         XCDB              ;CONVERT 2ND NUMBER
           JL ERROR        ;NO NUMBER
           JEQ ERROR       ;ONLY 2 PARAMETERS
         MOV R1,R0         ;OK, SWAP R1,R3
         MOV R3,R1
         MOV R0,R1
*
CONT     ....              ;R1=1ST, R3=2ND

PTR      DATA PTRS         ;STRING POINTER
DFP2     DATA 100          ;2ND PARAMETER DEFAULT
```

5.5.5 XGNP - GET NEXT PARAMETER

```
      Mnemonic:    XGNP
         Value:    >2FD0

        Format:    XGNP
                     L => No parameter
                    EQ => Null
                     H => parameter

     Registers: OUT (R1) = parameter
```

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 buffer (MIOP) and a parameter delimiter (MDEL).

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

A LOW status is returned if the last parameter delimiter is a null or period. XGNP does not parse past a period. In this case, register R1 is returned with a zero.

An EQUAL status is returned if the last parameter delimiter is a comma and no parameter follows. Register R1 is returned pointing to a null string.

A HIGH status is returned if a valid parameter is found. Register R1 then points to the parameter.

Possible Errors: None

```
SPAC    MOV @FDL(9),R0  ;GET SYSTEM DISK #
        SRL R0,8        ;POSITION
        XGNP            ;GET PARAMETER, OK?
          JLE SPAC02    ;N, USE DEFAULT
        XCDB            ;Y, CONVERT, OK?
          JLE ERR67     ;N, ERROR
        MOV R1,R0       ;Y
*
SPAC02  XSZF            ;GET DISK SIZE
          XERR          ;PROBLEM
        ....
```

```
.ASM SOURCE,BIN LIST ERR.SP
.CT (ASM SOURCE,BIN),15,,3
.DO ((DO DO),DO)
```

```
.LS.LS
```

```
.ASM SOURCE,,,ERR
```